МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гомельский государственный технический университет

имени П.О.Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

направление специальности 1-40 05 01-12 Информационные системы и технологии (в игровой индустрии)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине «Объектно-ориентированное программирование»

на тему: «Игровое приложение «Ферма» с использованием графики *DirectX*»

Исполнитель: студент группы ИТИ-21

Дубовцов И.Д.

Руководитель: преподаватель-стажер

Малиновский И.Л.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2022

СОДЕРЖАНИЕ

[Введение 4](#_Toc104420716)

[1 Программное обеспечение для разработки игрового приложения 5](#_Toc104420717)

[1.1 Основные подходы к созданию игр 5](#_Toc104420718)

[1.2 Технология *DirectX* 6](#_Toc104420719)

[1.3 Технология *OpenGL* 9](#_Toc104420720)

[1.4 *DirectX* 11 и *OpenGL* 4.0 11](#_Toc104420721)

[1.5 Необходимые условия для реализации игрового приложения 13](#_Toc104420722)

[2 Архитектура игрового приложения «Ферма» 14](#_Toc104420723)

[2.1 Архитектура игрового приложения. 14](#_Toc104420724)

[2.2 Средства реализации приложения «Ферма». Паттерны проектирования «Декоратор» и «Фабричный метод». 15](#_Toc104420725)

[2.3 Структура данных и схемы классов в проектах игрового приложения «Ферма». 16](#_Toc104420726)

[3 Результат верификации игрового приложения «Ферма» 21](#_Toc104420727)

[3.1 Принцип работы игрового приложения «Ферма» 21](#_Toc104420728)

[3.2 Тесты игрового приложения «Ферма» 23](#_Toc104420729)

[3.3 Результат верификации приложения 25](#_Toc104420730)

[Заключение 30](#_Toc104420731)

[Список использованных источников 31](#_Toc104420732)

[Приложение А. Листинг программы 32](#_Toc104420733)

[Приложение Б. Схема архитектуры приложения 89](#_Toc104420734)

[Приложение В. Руководство пользователя 90](#_Toc104420735)

[Приложение Г. Руководство программиста 92](#_Toc104420736)

[Приложение Д. Руководство системного программиста 93](#_Toc104420737)

# **ВВЕДЕНИЕ**

Популяризация компьютерной игровой индустрии начала свое зарождение с обычной идеи создания игры, в которой игрок перемешает по экрану световые лучи, возникнувшей еще до начала 1970-х годов. В это время Взаимодействие человека и вычислительной техники уже стало непреложным фактором.

Видео игры проникли во много сфер современной жизни. На сегодняшний день индустрия компьютерных игр является одной из самых прибыльных и скоростных по развитию отраслей компьютерных технологий и, вместе с этим, глобального сектора развлечений. Рынок компьютерных игр становится все глубже и уже к 2019 году вырос в среднем на 11% и обошел ряд конкурентных рынков, к примеру, кино-, музыкальная индустрия.

В этом проекте будет разработана игра жанра Аркада. Главная же особенность такого жанра – упрощенный игровой процесс и графика. Изначально аркадами были названы игры на игровых автоматах. Этим фактом и обусловлена их специфика: автоматы не подходили для сложных сюжетных игр, их ставили в местах большого скопления людей, поэтому они должны были быть привлекательными для широкой целевой аудитории, включая и детей, и взрослых, а соответственно, максимально простыми. Таким образом, разрабатывая игру жанра аркада, можно сказать, что будет разработана казуальная игра.

Казуальная игра содержит простые правила и простое управление. В таких играх нет необходимости обладать особыми навыками и располагать большим количеством времени. Основная аудитория казуальных игр – профессиональные геймеры и фанаты мобильного гейминга. Более 50 процентов пользователей мобильных приложений играют в игры, что уступает только приложениям для социальных сетей с точки зрения затраченного времени [1]. Казуальные игры живут не долго, но хорошо зарабатывают на показе рекламы. Казуальные игроки устанавливают в среднем в десять раз больше приложений в целом и в пять раз больше приложений со внутриигровыми покупками, чем пользователи других жанров [1]. Игра *Fall Guys* является свежим примером казуальной игры. Она стала популярна за пределами мобильной платформы и продалась тиражом в 10 миллионов копий, что является отличным результатом.

Для создания игр разработчики, художники и дизайнеры используют самые разные программы. И у каждого они свои. Как и сами технологии, все эти приложения постоянно меняются под влиянием потребностей и амбиций разработчиков. В курсовой работе автор раскроет поставленную тему и разработает игровое приложение. Для написания кода использован язык программирования *С#,* среда разработки *Visual Studio 2022*. Для решения задачи использован метод объектно-ориентированного подхода, свободная интегрированная среда разработки *SharpDX*, которая поддерживает разработку на *C#.*

1. **ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ РАЗРАБОТКИ ИГРОВОГО ПРИЛОЖЕНИЯ**

## **1.1 Основные подходы к созданию игр**

Можно выделить два основных подхода в реализации решения подобных задач – с использованием игрового движка и без него.

Игровой движок (англ. *game* *engine*) – это центральный программный компонент компьютерных и видеоигр, и других интерактивных приложений с графикой, обрабатываемой в реальном времени. Одним из ключевых преимуществ использования игрового движка, очевидно, является простота в использовании и, как следствие, реализации поставленной задачи. Перечень самых популярных игровых движков на сегодняшний день выглядит так:

*– Unity* – мультиплатформенный движок, который подходит для решения широкого спектра задач;

*– Unreal Engine* – один из самых популярных игровых движков на сегодняшний день. В связи с использованием С++ имеет огромнейших спектр возможностей и, в том числе, собственную визуальную систему программирования – *Blueprint*.

*– Construct* 2 – предназначен для эффективного и быстрого создания прототипов 2D-игр без помощи кода.

*– Corona* – кроссплатформенный движок, который поддерживает *iOS*, *Android*, *Microsoft Windows* и *Mac* с языком программирования *Lua* с недавнего времени стал полностью бесплатным.

*– Defold* – кроссплатформенный движок от компании King. Движок предназначен по большей части для работы с *2D*-проектами, но также поддерживает импорт *3D*-мешей. Скриптинг осуществляется посредством *Lua*.

Многообразие игровых движков позволяет нам найти такой вариант, который будет включать в себя необходимый нам функционал и будет удобен в использовании в нашем конкретном случае. С другой стороны, как бы усердно мы не подошли к выбору движка, мы все равно будем довольствоваться тем, что он может нам предложить, иначе говоря, мы будем ограничены лишь непосредственно его функционалом.

Подход к созданию игры, который не предполагает использование какого-либо движка, очевидно, является более сложным с точки зрения реализации поставленной задачи и предполагает наличие опыта и соответствующих навыков, но, несмотря на это, также имеет значительные достоинства, которые нельзя оставить без внимания. Существует большое количество языков программирования, с помощью которых можно разработать компьютерную игру: *С/С++,* *ActionScript*, *Java/JavaScript*, *Pascal*, *Basic*, *Delphi* и многие другие. Возможности графических библиотек языков программирования, как правило, широки. Более того, разработчик не ограничен функционалом движка, что в совокупности дает большую свободу действий с точки зрения реализации проектов.

## **1.2 Технология *DirectX***

До появления *DirectX* производители игр сражались с проблемами, которые связанны с несовместимостью оборудования. В связи с чем появилась проблема запуска одних и тех же игровых приложений из-за внушительного количества существующих конфигураций оборудования.

Индустрия столкнулась проблемой мультиплатформенных игр (стандартизации).

В ответ на проблему *Microsoft* представила *Windows Game SDK* для *Windows 95*, чем, собственно, и был *DirectX* 1. С возможностями *DirectX* 1 разработчики игровых приложений получили общий набор *API*, который в большинстве случаев предоставлял гарантию совместимости с различными наборами аппаратного обеспечения ПК. К тому, что с момента релиза *DirectX* количество игр, разработанных под платформу Windows игр, динамически увеличивалось стоит относиться спокойно. Такая статистика была актуальна 15 лет [2, c. 1].

*DirectX* – набор *API* (интерфейсов прикладного программирования) *Microsoft*, предназначенный для использования разработчиками низкоуровневого интерфейса, сопоставленного с аппаратным обеспечением ПК для операционных систем базированных на *Windows*. Каждый его компонент аппаратного обеспечения, который включает графику, звук, общие вычисления на графическом процессоре и устройства ввода, разработан под одни стандартный интерфейс [2, c. 2].

*DirectX* основан на идее графического конвейера. Программист может настроить отдельные этапы этого конвейера и затем подавать на него входные данные. Структура *DirectX* представлена на рисунке 1.1.

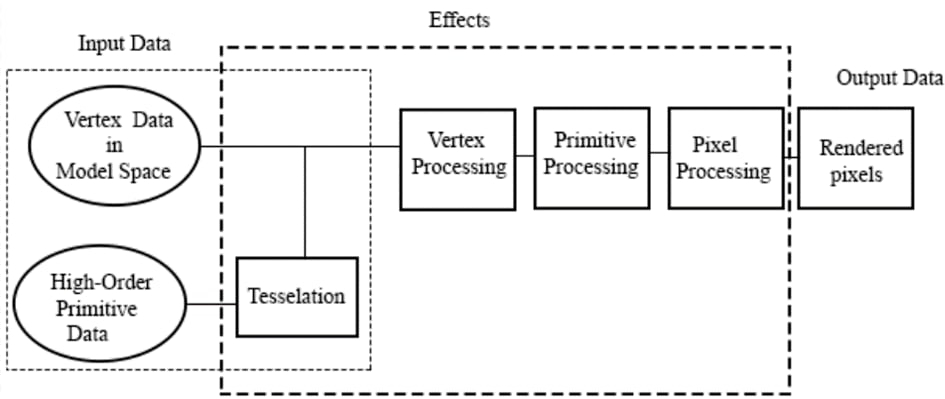


Рисунок 1.1 – Архитектура *DirectX*

Такая идеология позволяет разработчиками писать игровые приложения используя стандартный *API*, не беспокоясь о разных низкоуровневых взаимодействиях с различными аппаратными средствами на стороне клиента. Подумать страшно о том, как разработчики пишут разные пути к коду для множества различных конфигураций оборудования, которые могут быть у игрока. Как правильно принять решение, если у пользователя было друге устройство ввода? Другая операционная система? Что делать с возможно различными драйверами? Разными типами графического оборудования?

Именно наличие одного стандартного *API*, к которому придерживаются все производители оборудования является идеальным решением, чем индивидуальная поправка под каждого клиента на рынке.

Таким образом *DirectX* – набор стандартных *API*-интерфейсов, которые используются в основном разработчиками видеоигр для стандартизации разработки под платформы *Windows* и *Xbox*. Производители оборудования должным образам задают набор драйверов для своих устройств.

*DirectX API* разделен на несколько компонентов, каждый из который представляет из себя часть другой системы. Каждый *API* может использоваться независимо от другого. Такой подход позволяет избавиться от лишнего функционала в игре. В новейших версиях *DirectX* некоторые компоненты были обновлены, такие как *Direct3D*, в то время как другие части *API* поддерживаются Microsoft на тех же уровнях.

В этой работе *Direct3D* будет рассматриваться посредственно. Это напрямую связанно с поставленной задачей.

Значит, рассмотрим эти компоненты [2, c. 7].

*Direct2D* используется для *2D* графики под тридцати двух битные приложения. Способен отображать векторную графику с высокой производительностью.

*DirectWrite* используется для отображения шрифтов и текста в Direct2D приложении.

Компонент *DXGI* является графической инфраструктурой *DirectX*, также известен в широких кругах как *DXGI*. В основном, используется для создания очереди смены изображений.

*XAudio2* – это низкоуровневый *API* для обработки звука, который является частью *XDK* (*Xbox Development Kit*), а сейчас *DirectX SDK*. *XAudio2* это замена *DirectSound*. В свою очередь, оригинальный *XAudio* использовался ещё на первых консолях *Xbox*.

*XInput* – это *API* ввода для *XDK* и *DirectX SDK* и используется для обработки ввода со всех *Xbox 360* контроллеров. Любые контроллеры можно использовать с *Xbox360*, которые также будут применяться к ПК. В этом случае, *XInput* – это *API*, который используется для работы с этими устройствами. Под устройствами имеется ввиду не только геймпады *Xbox*, но и также *Rock Band* и Guitar Hero контроллеры. *XInput* является заменой *DirectInput*.

*XNA Math* – это не *API*, а скорее просто математическая библиотека, которая реализует оптимизированные математические операции, которые характерные для видеоигр. *XNA Math* использует *SIMD* (*Single Instruction Multiple Data*) для выполнения нескольких операций с помощью одного вызова команды. Математическая библиотека *XNA* доступна для *Xbox 360* и ПК с *Windows*.

*DirectCompute* является достаточно свежим *API* добавленным в *DirectX*, позволяющий выполнять многопоточные вычисления общего назначения с использованием графического процессора. Графические процессор имеет возможность обрабатывать множество задач параллельно, такие как физические, сжатия и распаковки видео, аудио обработки и другие. Не все задачи подходят для графического процессора, но, для вышеперечисленных, возможности огромны.

Допустим, что игра готова, теперь имеется желание показать её остальным. *DirectSetup* открывает функционал для скачивания последней версии *DirectX* на устройство пользователей. Также имеется возможность проверки скаченной версии *DirectX*.

*Game Explorer* является функцией *Windows Vista* и *Windows 7*, который позволяет разработчиками презентовать свои игры на этих операционных системах. Обозреватель игр обрабатывает такие вещи как отображение игры, название, рейтинг, описание, обложка для определенного региона, рейтинги контента (к примеру, M для взрослых, T для подростков), статистика игры и уведомления, родительский контент и многое другое. *DirectX SDK* обеспечивает большим количеством информации о том, как использовать *Game Explorer* для своих игр и будет полезным когда придет время рассылать игру.

*DirectInput* – это *API* для обнаружения ввода и помощью клавиатур, мышей и джойстиков.

На сегодняшний день *XInput* используется для всех игровых контроллеров. Для клавиатур и мышек можно использовать *Win32* функции или использовать *DirectInput*.

Следующие, перечисленные автором, компоненты либо устарели, либо удалены из *DirectX SDK* [2, c. 11].

*DirectDraw*, когда-то использовавшийся для *2D-*рендеринга. Сегодня можно использовать либо *Direct2d*, либо *Direct3D* для *2D*-графики. *DirectDraw* был объединен с *Direct3D* в то, что в *DirectX 8* называлось *DirectX Graphics*.

*DirectPlay* использовался для создания сетевых возможностей для онлайн-игр. Он был построен поверх протокола *UDP* (*User Datagram Protocol*) и служил абстракцией более высокого уровня для сетевого взаимодействия. Сегодня *API* был удален и больше не является частью *DirectX SDK*. *DirectPlay* устарел для *Windows Live* на ПК и *Xbox Live* на консолях *Xbox*.

*DirectShow* использовался для рендеринга и записи мультимедиа. Это означало, что *DirectShow* имел возможность отображать видеофайлы в нескольких распространенных форматах, обеспечивать навигацию по *DVD* и много другое. На сегодняшний день *DirectShow* является частью *Windows SDK*, и больше и не является частью *DirectX SDK*. Кроме того, пользователи *Windows Vista* и *Windows 7* могут использовать *Microsoft Media Foundations* для мультимедийного контента, который также является частью *Windows SDK*. Это полезно для видеоигр, если необходимо отображать кат-сцены *CG* и видеофайлы.

*DirectMusic* на данный момент является устаревшим *API DirectX 7* и более ранних версий для воспроизведения аудио контента в приложениях. *DirectMusic* предполагает низкоуровневый доступ к аудио и аппаратному обеспечению. Уже много лет является темой книг и руководств по *DirectX*. Сегодня используется *XAudio2* (низкоуровневый) для воспроизведения звука в играх и мультимедийных приложениях.

*DirectSound* – ещё один устаревший аудио *API*, используемый для предоставления разработчиками низкоуровневого доступа для их аудио потребностей. *XAudio2* является очень даже приемлемой заменой.

Перечисляя эти компоненты, автор обосновывает, почему они небыли использованы в работе.

*DirectX API* базируется на компонентной объектной модели (*Components* *Object Model* – *COM*).

*COM*-объекты состоят из коллекции интерфейсов, который представляют методы, что используют разработчики для доступа к *DirectX*. *COM*-объекты – это обычные *DDL*-файлы, которые были зарегистрированы с системой для обеспечения поддержки определенного оборудования на компьютере. *COM* регистрация для объектов *DirectX* происходит во время установки *DirectX*. Будучи похожими на объекты *C++*, *COM*-объекты требуют использования интерфейса для доступа к методам внутри них, что, на самом деле, является преимуществом перед стандартными объектами, потому что в *COM*-объекте может присутствовать несколько версий интерфейса, что обеспечивает обратную совместимость.

К примеру, каждая версия *DirectX* включала новый интерфейс *DirectDraw*, доступный через *API* одновременно сохраняя предыдущую версию, чтобы не нарушить существующий код. Таким образом игры, созданные с использованием *DirectX 7*, могу работать с *DirectX 9*. Другими словами, старые игры можно устанавливать и воспроизводить с использованием последней версии среды исполнения *DirectX*.

Дополнительным плюсом *COM*-объектов является их способность работать с разными языками программирования, не только *C++*. Разработчики могут использовать *Visual Basic*, *C++*, или *C#* и также использовать те же библиотеки *DirectX*. Вместе с обновлениями Microsoft и добавлением нового функционала в *DirectX*, версия каждой компоненты будет увеличиваться. Для примера, *DirectInput* остался на версии 8.0, в то время как *Direct3D* имеет версию 11 [2, c. 11].

## **1.3 Технология *OpenGL***

*Open Graphics Library* – открытая графическая библиотека, которая, на данный момент, является одним из самых популярных прикладных программных интерфейсов *API* для разработки приложений в области двумерной и трехмерной графики. Данную библиотеку разрабатывает некоммерческая организация *Khronos Group* совместно с её сообществом. Большее количество производителей *GPU* (*Graphics Processing Unit*), в той или иной мере, вносили вклад в реализацию *OpenGL*. Данный стандарт доступен на большинстве современных платформ, что позволяет ему конкурировать с ранее описанной технологией *DirectX*. *OpenGL* является основным компонентом при взаимодействии с *GPU* в операционных системах *Mac* и *Linux*. Данный продукт можно считать промежуточной частью между пользовательским уровнем и аппаратной частью, который предоставляет программистам единый интерфейс для различных платформ и использует для аппаратной поддержки.

На рисунке 1.2 представлена графическая схема конвейерной архитектуры *OpenGL*:

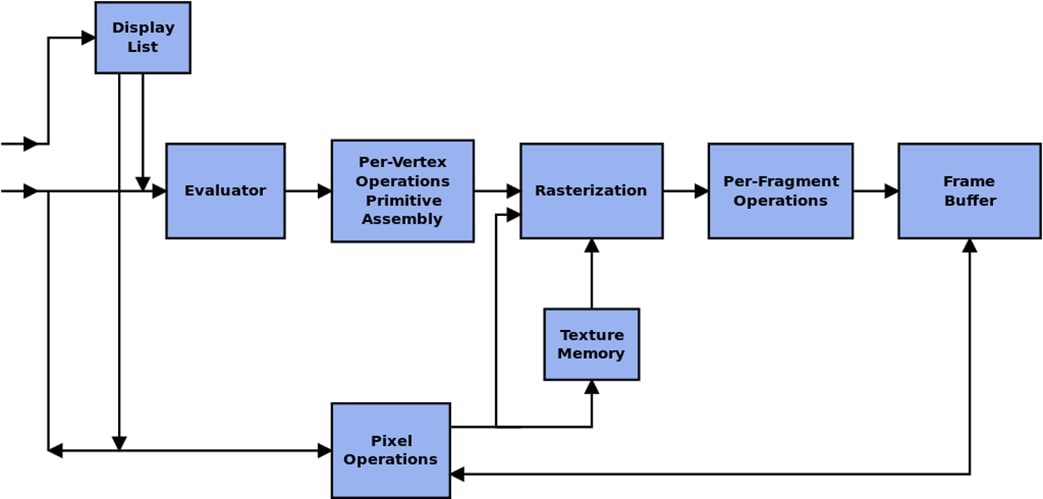


Рисунок 1.2 – Конвейерная архитектура *OpenGL*

Реализация *OpenGL* для работы графического приложения использует стандартные функции языка программирования *С*. Для данных функций существует стандартизированный *ABI* (*Application Binary Interface*). Это означает, что возможно использование *OpenGL* из любого языка программирования, который имеет поддержку вызовов функций из *native* библиотек.

В основе большинства возможностей *OpenGL* лежит машина состояний, а результат вызовов функций *OpenGL* зависит от её внутреннего состояния, которое способно изменить контекст графического приложения. Для того, чтобы получить возможность манипулировать конкретным объектом в *OpenGL*, для начала требуется выбрать объект в качестве текущего, и затем его можно будет изменять.

Библиотека утилит *OpenGL* (*OpenGL Utility Library - GLU*) предоставляет множество средств для моделирования, например, квадратические поверхности, кривые и поверхности типа *NURBS*. *GLU* – стандартная часть любой реализации *OpenGL*. Существуют также и более высокоуровневые библиотеки, например, *Fahrenheit Scene Graph* (*FSG*), которые построены с использованием *OpenGL* и распространяются отдельно для многих ее реализаций.

В некоторых реализациях (например, в реализации для системы *X Window*), *OpenGL* разработана таким образом, чтобы работать со многими машинами. Это может происходить в случае, если работа происходит в сетевом окружении, состоящем из множества компьютеров, соединенных между собой. В данной ситуации компьютер, на котором функционирует программа, вызываются команды *OpenGL*, является клиентом, в то время как компьютер, осуществляющий отображение, является сервером.

Реализации *OpenGL* разрешают пользователям добавлять различные модификации к основной спецификации, что в свою очередь позволяет приложению получить доступ к списку поддерживаемых *OpenGL* расширений прямо во время выполнения приложения, а также проверить на доступность те расширения, которые приложение будет использовать. Большая часть функционала *OpenGL* – это различные расширения. Такой подход позволяет приложениям пользователей использовать инструменты, которые производитель только недавно реализовал в своем драйвере и задокументировал, не дожидаясь включения данных возможностей в официальную спецификацию. Самые успешные расширения со временем могут стать основами спецификации *OpenGL*.

Поэтому каждая новая версия *OpenGL* – это сет новых интегрированных расширений для более старой версии. Данный подход позволяет программисту не придавать значения версии *OpenGL*, которую он использует, так как для программы главное лишь доступные ей расширения.

## **1.4 *DirectX* 11 и *OpenGL* 4.0**

*OpenGL* долгое время считался конкурентноспособным и конкурирующим графическим *API* по отношению к *Direсt3D*. *OpenGL* поддерживает платформы за пределами операционных систем на базе *Windows*, а это *Mac*, операционные системы *Linux*, *iPhone от Apple*, *PlayStation 3* от *Sony* (реализация), также множество мобильных устройств, таких как сотовые телефоны. Хотя создание собственного устройства *OpenGL* может отличаться от одной платформы к другой, остальная часть *API* считается независимой от платформы, не учитывая долгую историю аппаратного обеспечения конкретных расширений и функций от конкурирующих организаций в рамках *OpenGL*. C другой стороны, *DirectX* доступен на различных операционных системах *Windows* и игровых консолях *Xbox*. Чтобы cчитать совместимым с *DirectX* *10* или 11, оборудование должно соответствовать строгому списку совместимости, но до *DirectX 10* это было не всегда так. В случае c *OpenGL* это часто приводило к специфичным для поставщика расширениям, которые работали только на определенном оборудовании.

При попытке добиться одинакового результата на всех поддерживаемых устройствах, несовместимость на рынке аппаратного обеспечения привела к переработке.

Вся дискуссия про *Direct3D* и *OpenGL* часто может показаться религиозной, но факт в том, что в течение многих лет *OpenGL* оставался позади Direct3D.

Компания *Microsoft* проделала огромную работу по разработке *Direct3D* и улучшению его на протяжении многих лет, а *OpenGL* только отставал, не выполняя своих обещаний по выпуску каждой новой версии и также страдал от тех же проблем минувшего года. Когда *OpenGL 3.0* был анонсирован считалось, что *OpenGL*, вернется в положение конкурента *Direct3D*. К сожалению, на долю *OpenGL* пришли как взлеты, так и падения, как внутри группы, отвечающей за разработку, так и вместе с тем, как *API* сочетался с *Direct3D*. *Direct3D* продолжал доминировать.

*OpenGL 4.0* стал догонять многие рекламированные функции *DirectX 11*, в свою очередь благодаря поддержке вычислений общего назначения с использованием *OpenCL* и тесселяции. Начал позиционировать себя на шаг ближе к обещаниям, которые небыли выполнены в прошедших годах, особенно с *OpenGL 2.0* и *OpenGL 3.0* [2, c. 5].

Таким образом, стоит обобщить некоторые важные достоинства *DirectX* и *OpenGL*:

a)*DirectX*;

1) собственный стандарт *Microsoft* (2.0 - 11.1);

2) поддерживается производителями аппаратного обеспечения;

3) обеспечивает взаимодействие с устройствами;

4) первоначально ориентирован на применение разработчиками игр;

5) преимущественно ОС *Windows*;

б) *OpenGL*;

1) открытые стандарты *OpenGL* (1.1 - 4.3), *GLSL* (1.10 - 4.30), *OpenGL ES* (1.0, 1.1, 2.0, 3.0);

2) поддерживается производителями аппаратного обеспечения, позволяет иcпользовать расширения;

3) полная абстракция от платформы и ОС;

4) первоначально ориентирован на профессиональное применение;

5) ОС: *Windows* (*OpenGL* 1.1 в случае отсутствия драйверов), *Linux*, *MacOS*, *iOS*, *Android*, *Symbian*.

На данный момент, сказать какое *API* является более востребованным –тяжелое решение. В этом случае вывод можно свести к изучаемой предметной области (к примеру, в данном проекте автору выгодно использовать *DirectX*, так как приложение будет разрабатываться на *Windows* и *DirectX* оптимизирован для разработчиков игр), однако, важно отметить, *DirectX* по своей сути фиксирован в пределах одной мажорной версии. Это говорит о том, что какие-либо дополнения приходят только при выпуске новой версии. У *OpenGL* практически весь функционал – это расширения, то есть процесс развития можно описать так: появляется идея – разработчик реализовывает – новая функция уже может быть использована.

## **1.5 Необходимые условия для реализации игрового приложения**

Для разработки игрового приложения, понадобится среда разработки. Как оговаривалось ранее, будет использоваться *Visual Studio*, так как эта среда разработки, в первую очередь, располагает ресурсами для разработки приложений и игр, также имеет бесплатный пакетный менеджер *NuGet* с открытым исходным кодом, служащий для .*NET* механизмом совместного использования кода [7]. Во вторую очередь, *Visual Studio* предоставляет возможность быстрой навигации и исправления кода, простой отладки и диагностики, имеет комплексные инструменты тестирования [5].

Для создания спрайтов используется *Adobe Photoshop CC 2019* исходя из предпочтений автора.

Одной из характерных особенностей программного обеспечения такого приложения является то, что для использования графической библиотеки *DirectX* в паре с языком программирования *C#* необходимо использовать низкоуровневую оболочку *SharpDX*, так как большинство ресурсов, доступных пользователю, в основном предназначены для *С++,* а не для *C#.* Данная оболочка позволяет решить проблему использования языка *C#* и технологии *DirectX*.

Исходя из темы (игровое приложение **«**ФЕРМА**»** с использованием графики *DirectX*), для работы с *DirectX*, воспользуемся возможностями встроенного в *Visual Studio* пакетного менеджера *NuGet* и установим следующие библиотеки:

*– SharpDX*;

*– SharpDX.Direct2D1*;

*– SharpDX.Mathematics*;

*– SharpDX.Desktop*;

*– SharpDX.DXGI*;

*– SharpDX.DirectInput*.

Далее используя инструкцию *using* подключить их.

1. **АРХИТЕКТУРА ИГРОВОГО ПРИЛОЖЕНИЯ «ФЕРМА»**

## **2.1 Архитектура игрового приложения.**

Для проектирования архитектуры игрового приложения был использован метод декомпозиции. Это универсальный научный метод решения сложных задач, основанный на использовании структуры задач и ее разбиении на более простые подзадачи. Полученное решение проектировалось и разрабатывалось.

Схема архитектуры спроектированного приложения представлена в приложении Б.

Чтобы реализовать игровую логику, необходимо создать игровой движок.

Игровой движок будем расценивать как проект, позволяющий разделить игру на определенные компоненты:

– обработка ввода с клавиатуры;

– игровые объекты;

– окно для отображения;

– поведение на сцене;

– отображение графики.

Библиотека игрового движка предполагает собой определенный алгоритм работы. В данном случае, алгоритм будет отвечать за подготовку ресурсов для корректной работы приложения, циклического рендеринга (воспроизведения и отрисовки) объектов на протяжении работы приложения (всей игры), также за высвобождение ресурсов перед закрытием. Контроллеры, которые содержат функционал для обработки нажатий, который отвечает за управление игроком и инвентарем.

Таким образом, поведение объекта может быть изменено с помощью добавления или удаления компонентов. Такой подход позволит решить проблемы неоднозначности глубокой и широкой иерархии наследования, трудные для поддержки

Библиотека игровой логики содержит основные игровые объекты, которые используются при инициализации игровой сцены. Данная библиотека будет содержать сущности:

­ – конструктор игрока;

– фабрики для создания инструментов, эффектов, растений;

– инвентаря игрока, позволяя добавлять элементы взаимодействия для игрока.

Библиотека интерфейса приложения отвечает за отображения окон. В этой библиотеке будет находится класс программы.

Процесс декомпозиции позволил разработать обобщённый алгоритм работы проекта игровой логики. Ключевым элементом в алгоритме является сцена, которая создает, отображает, динамически добавляет и удаляет элементы в игре. Также сцена отслеживает конец игры. В алгоритме каждый блок действий представляет собой этап разработки, который разбивается на подзадачи.

Каждый из этапов разработки позволяет получить всю необходимую информацию о ресурсах и действиях. После того, как информация собрана можно точно осознать и обозначить цель, и продумать последовательность подзадач для её достижения. Также легко сформировать структуру приложения.

## **2.2 Средства реализации приложения «ФЕРМА». Паттерны проектирования «Декоратор» и «Фабричный метод».**

В игре «ФЕРМА» часто придется расширять функционал объектов семян, если точнее, то параметры роста. В таком случае, нужно создать класс базового декоратора. С его помощью мы получим механизм подключения компонента и обеспечим переадресацию методов и свойств. Далее, с помощью наследования можно создать классы конкретных декораторов, при этом реализуя в них только свойства и метода с измененным функционалом. Как результат, в ходе разработки мы сможем получать различные объекты, которые будут отличаться только параметрами [4].

На рисунке 2.3 представлена схема реализации паттерна «декоратор» для изменения процесса выращивания растений.

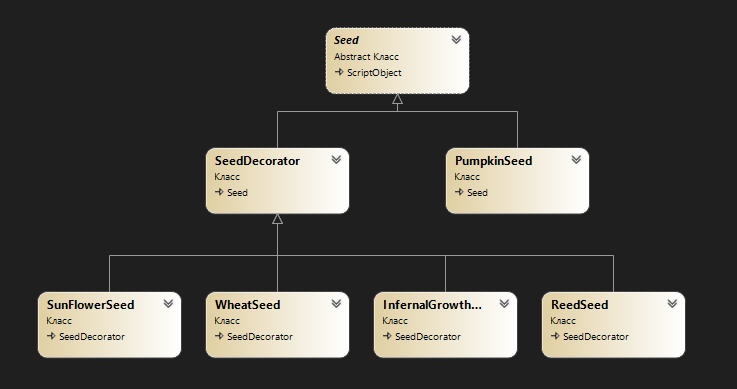


Рисунок 2.3 – Схема реализации паттерна «декоратор»

Выделим преимущества и недостатки паттерна «декоратор» [6, c. 102]:

а) преимущества:

– высокая гибкость, по сравнению с наследованием;

– динамическое придание свойств объекту;

– множественное придание свойств одному объекту;

б) недостатки:

– усложненная конфигурация объектов при многократном «обертывании» одного и того же объекта;

– множество небольших классов.

В игре часто придется удалять и создавать объекты динамически по ходу выполнения программы. Для этого нужно реализовать паттерн «фабричный метод». Суть в том, что он определяет интерфейс для создания объекта, то есть является порождающим шаблоном проектирования, но оставляет подклассам задачу о создании конкретного класса. Суть механизма, который использует этот паттерн, заключается в том, что используется форма полиморфизма, то есть классы всех конечных типов будут наследоваться от одного базового класса.

По ходу игры игрок будет садить семена, что требует постоянного создания объектов на сцене. Значит, необходимо создать абстрактные фабрики для каждого из объектов. Такая структура позволит легко менять, вносить изменения в проект.

На рисунке 2.4 представлена схема реализации шаблона проектирования «фабричный метод» для создания растений различных типов.

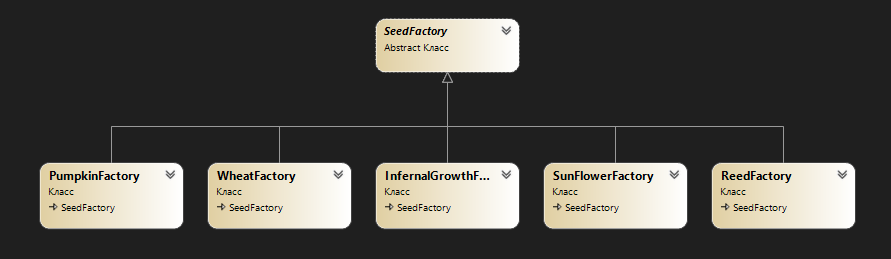


Рисунок 2.4 – Схема реализации паттерна «фабричный метод»

## **2.4 Структура данных и схемы классов в проектах игрового приложения «ФЕРМА».**

В этом подразделе будут подробно рассматриваться созданные проекты игрового приложения, и то, что они из себя представляют.

Игровой движок содержится в проекте с названием «*EngineLib*».

В нем имеются следующие классы:

а) компоненты движка:

*– ApplicationRendering*;

*– Input*;

*– InputHandler*;

*– RenderSystem*;

*– Scene*;

*– Time;*

б) компоненты игрового объекта:

*– Collider*;

*– GameObject*;

*– ScriptObject*;

*– Sprite*;

*– TextElement*;

*– Transform*.

Теперь рассмотрим каждый класс.

*ApplicationRendering* – один из основных классов проекта *EngineLib*.

В его конструкторе создается окно (*RenderForm*) и инициализируется обработчик нажатий клавиш, свойства окна (режим переднего плана, начальная позиция, высота ширина, гибкость окна).

*RenderSystem* – также является одним из основных компонентов проекта. Отвечает за инициализацию свойств полотна (целевого окна рендеринга). Этот класс имеет метод *LoadBitmap*, который конвертирует картинку в тип данных *Direct2D1.Bitmap* и на выходе отдает готовую картинку для загрузки.

*InputHandler* – инициализирует состояние и доступ к клавиатуре пользователя.

*Input* – содержит специфичные методы, которые обрабатывают нажатие клавиш, в случае если отработал *InputHandler*.

*Scene* – является абстрактным классом, который содержит в себе списки объектов: текущие объекты, объекты для добавления в следующем кадре, объекты для удаления в следующем кадре.

Также, содержит методы:

*– CreateScenesGameObjects* – является абстрактным методом, которые инициализирует сцену игровых объектов до запуска рендеринга;

*– DrawScene* – метод, который рендерит игровые объекты;

*– RemoveObjectsFromScene* – метод удаления игровых объектов, исходя из списка объектов, которые нужно удалить в следующем кадре;

*– AddObjectsToScene* – метод добавления игровых объектов, исходя из списка объектов, которые нужно добавить в следующем кадре.;

*– EndScene* – виртуальный метод, который изменяет статус отрисовки сцены.

*Time* – класс, который служит для отслеживания времени существования приложения. Содержит метод *Reset* для обнуления таймера.

*GameObject* – класс, который описывает все сущности в проекте. Содержит метод *Draw*, который вызывается при каждой смене кадра, который рисует спрайт игрового объекта.

*Сollider* – отслеживает взаимодействие игровых объектов друг с другом при прикосновении. Содержит метод *UpdateBounds*, который обновляет границы зоны взаимодействия, перегруженный метод *CheckIntersection*, который реагирует на коллизии всех игровых объектов, у которых есть коллайдер.

*ScriptObject* – является абстрактным классом, который представляет из себя сценарий игрового объекта. Имеет методы:

*– Initialize* – метод для инициализации скрипта игровым объектом;

*– Start* – абстрактный метод, который представляет из себя поведение игрового объекта на момент создания;

*– Update* – абстрактный метод, который обновляет состояние объекта.

*Sprite* – класс, который содержит спрайт игрового объекта.

*TextElement* – класс, который отображает текст, что привязан к игровому объекту. Содержит метод *SetText*, который обновляет текстовую строку объекта.

*Transform* – класс, описывающий положение и размер объекта на сцене, имеет метод *SetMovement*, который обновляет положение объекта.

Отсюда можно сделать вывод, что в проекте *EngineLib* решена задача создания игровых объектов и сцены. Также реализован класс *ScriptObject*, который реализует логику игры.

Схема проекта *EngineLib* продемонстрирована на рисунке 2.5.



Рисунок 2.5 – Схема проекта *EngineLib*

Проект *GameLib* содержит следующие классы:

*– Fermas* – класс, который описывает игровое пространство и сцену;

*– ChangingDay* – класс, наследуемый от класса *ScriptObject*. Отвечает за смену времени суток;

*– DelayTime* – класс, который отвечает за задержку между определенными действиями;

*– ObjectsFactory* – класс, который создает игровые объекты в момент инициализации сцены;

*– InventoryObjectsFactory* – класс, который, как и *ObjectsFactory* инициализирует сцену, но объектами с текстовой строкой;

*– PlayerConstructor* – класс, который отвечает за создание игрока и задание ему начальных свойств. Содержит метод *CreateTools*, который создает инструменты игрока;

*– Player* – класс унаследованный от *ScriptObject*. Отвечает за управление игровым объектом персонажа. Содержит информацию о предметах, которые находятся в инвентаре, количестве набранных монет. В методе *Update* обрабатывается передвижение игрока и обрабатываются коллизии со стенками;

*– Inventory* – класс, который, собственно, является инвентарем игрока. В нем содержатся словари с информацией о инструментах и семенах. Присутствует статический экземпляр, к которому обращаются сценарные объекты, чтобы воспользоваться методами;

*– GrowScript* – абстрактный класс, который наследуется от *ObjectScript*. Определяет и содержит необходимую логику для посадки определенных растений на определенной территории;

*– CellScript* – абстрактный класс, наследуемый от *ObjectScript*, который содержит всю необходимую логику для отображения информации об инвентаре. Содержит метод *Update*, в котором изменяется текстовая строка объекта, меняется спрайт определенного реализующего объекта в зависимости от выбора игрока;

– *MoneyCell* – реализует абстрактный класс *CellScript*. Отвечает за текст отображения количества набранных монет;

– *PumpkinCell* – реадизует абстрактный класс *CellScript*. Отвечает за текст, которые отображает количество семян тыквы.

*– Tool* – абстрактный класс, который содержит всю необходимую логику созданных инструментов. Содержит абстрактные методы *ToolDelayInit*, *ToolDelayReset*. Где первый задает задержку, а второй обновляет её;

*– BasketTool* – реализует абстрактный класс *Tool*. Его создание реализовано в классе *ToolFactory*;

– *Seed* – абстрактный класс, который содержит всю необходимую логику растений. Содержит метод *Update*, который отвечает за процесс роста.

*– PumpkinSeed* – реализует абстрактный класс *Seed*. Его создание реализовано с помощью паттерна «фабричный метод»;

*– ToolEffect* – абстрактный класс, который описывает общую логику игрового объекта. Содержит метод *Update*, в котором вызывается метод Interaction с индивидуальной реализацией каждого эффекта. Также содержит абстрактный метод *CheckActivate*, который отвечает за возможность активирования эффекта (это обусловлено тем, что эффект может иметь шанс для активации), абстрактные методы *ActivateEffect* и *DeactivateEffect*, которые отвечают за активирование эффекта, в случае если отработал метод *CheckActivate*, и деактивирование эффекта, отвечающее за отключение действия эффекта на инструмент;

*– WateringToolEffect* – класс, реализующий абстрактный класс *ToolEffect*. Ускоряет рост семян на один день, также с определенным шансом ускоряет игрока на заданное количество времени, уменьшает время задержки на сбор и посадку растений при активации. Его создание реализовано в классе *ToolEffectFactory*.

На рисунке 2.6 представлена схема проекта *GameLib*.

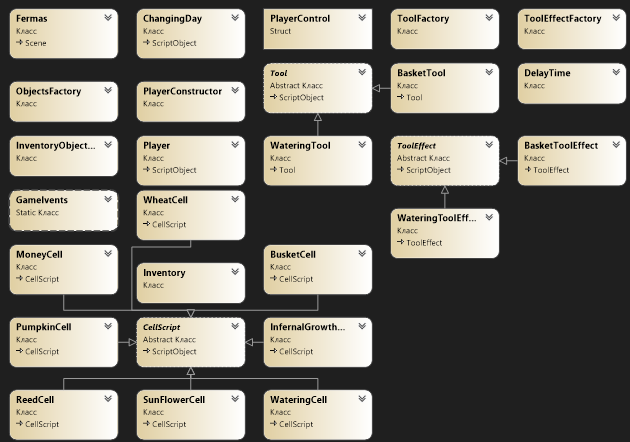


Рисунок 2.6 – Схема проекта *GameLib*

Событие закрытия игрового приложения осуществляется с помощью делегата, находящемся в статическом классе *GameEvents*. Реализовав такую гибкую структуру, можно легко отображать любую игровую статистику.

1. **РЕЗУЛЬТАТ ВЕРИФИКАЦИИ ИГРОВОГО ПРИЛОЖЕНИЯ «ФЕРМА»**

**3.1 Принцип работы игрового приложения «ФЕРМА»**

Перед запуском самой программы, в среду разработки загружаются необходимые библиотеки для работы с *DirectX*. При запуске игрового приложения открывается проект пользовательского интерфейса Windows Form. В классе *Form1* c с помощью класса *ApplicationsPicture* (Приложение А, листинг кода *ApplicationsPicture.cs*) картинками инициализируется окно интерфейса. Инициализируются компоненты интерфейса и создаются экземпляры классов игрового движка. Элементы *PictureBox*, *Label*, *Button* способствуют выводу необходимой информации для игр, а также запустить игру соответственно.

Нажатие кнопки «*Play*» в пользовательском интерфейсе инициализирует игровую сцену в классе *ApplicationRendering* (Приложение А, листинг кода *ApplicationRendering.cs*), там же создается экземпляр класса *RenderForm* (Приложение А, листинг кода *RenderForm.cs*). По ходу инициализации создается игровой персонаж и уровень. Игровое поле создается исходя из выбранного файла .*bmp*, который содержит структуру поля, где цвет каждого пикселя соответствует определенному элементу игры сцены. Игровые объекты создаются в классе *ObjectsFactory* (Приложение А, листинг кода *ObjectsFactory.cs*) и *InventoryObjectsFactory* (Приложение А, листинг кода *InventoryObjectsFactory.cs*). Также сцена содержит список пустых игровых элементов и хранит в себе методы удаления и добавления игрового объекта со сцены.

После того как сцена проинициализирована запускается зацикленная инфраструктура рендера, которая будет визуализировать каждый кадр игрового окна с помощью метода *RenderCallBack*. В этом методе каждый кадр будет обновлять состояние клавиатуры и времени игрового приложения. Каждый игровой объект должен быть проинициализирован компонентами движка, после чего отображаться в каждом кадре с помощью метода *Draw* до тех пор, пока игровой объект существует на сцене.

Без ввода с клавиатуры игры не может быть, как таковой, поэтому класс *InputHandler* (Приложение А, листинг кода *InputHandler.cs*)отвечает за отклик событий ввода. Экземпляр этого класса создается *ApplicationRendering*. C помощью метода *RenderCallBack* обновляется каждый кадр состояние клавиатуры. Нажатия клавиш отслеживаются с помощью свойства *KeyboardState*. Статический класс Input по своей сути упрощает процесс получения доступа к клавиатуре, но немного ограничивает функционал класса *InputHandler*.

Статический класс *Time* (Приложение А, листинг кода *Time.cs*) отвечает за работу со временем. Содержит методы текущего времени в данный кадр и времени последнего кадра. Каждый вызов метода *RenderCallback* обновляет значение свойств *CurrentTime* и *DeltaTime*. Благодаря *DeltaTime* в качестве множителя при вычитании или сложении значений каждого кадра делает приложение независимым от времени смены кадров.

После ввода с клавиатуры происходит передвижение, которое независимо от места расположения персонажа на игровом поле, происходит по вертикали и горизонтали. Если игрок сталкивается с забором, водой или лавой, то игрок не может продолжить передвижение. Также есть такие игровые объекты как земля, песок и песок душ, при столкновении с которыми игрок может двигаться, а также при выборе соответствующего объекта в инвентаре может выполнить заданное действие.

Игровое поле представляет из себя участки сценарных объектов *GrowScript* (Приложение А, листинг кода *GrowScript.cs*), таких как *LandScript* (Приложение А, листинг кода *LandScript.cs*), *ShowerSendScript* (Приложение А, листинг кода *ShowerSendScript.cs*), *SendScript* (Приложение А, листинг кода *SendScript.cs*), который унаследован от *ScriptObject* (Приложение А, листинг кода *ScriptObject.cs*). При самом старте игры класс *Inventory* игрока инициализируется рандомным количеством семян. В классе *GrowScript* проверяется пересечение объекта игрока с любым из объектов территории роста, после чего, в зависимости от выбранного игроком компонента инвентаря добавляется или удаляется игровой объект, количество определенного объекта в инвентаре либо добавляется, либо отбавляется, в зависимости от ситуации.

Игровые объекты, которые добавляет игрок, можно удалить только при достижении этими объектами заданного времени и используя инструмент «корзинка». За удаление объектов таким инструментом выдаются деньги в виде монет. Их количество отображается внизу игрового окна.

Игрок может выбрать, какие именно объекты хочет добавить на сцену, для этого ему нужно выбрать соответственную ячейку, в которой показано семя и его количество в инвентаре. В случае если у выбранной ячейки количество семян равно 0, то игрок не сможет добавить выбранный объект на сцену.

Также в игре присутствует ещё одни инструмент – «лейка». Сценарные объекты класса *Seed* (Приложение А, листинг кода *Seed.cs*), такие как *PumpkinSeed* (Приложение А, листинг кода *PumpkinSeed.cs*), *ReedSeed* (Приложение А, листинг кода *ReedSeed.cs*), *InfrernalGrowthSeed* (Приложение А, листинг кода *InfernalGrowthSeed.cs*), *WheatSeed* (Приложение А, листинг кода *WheatSeed.cs*), *SunFlowerSeed* (Приложение А, листинг кода *SunFlowerSeed.cs*), унаследованные от *ScriptObject*, в методе *Update* обновляют свое время роста, в случае если таймер не дошел до определенного количества времени, имеют статус «*grow*», если таймер уже пересек заданную отметку роста, то имеют статус «*ready*». Класс *WateringTool* (Приложение А, листинг кода *WateringTool.cs*), который представляет инструмент «лейка», следит за пересечением игрока и пересечением его с объектами роста. В случае возникновении коллизии, объект, с которым пересекается игрок и выбранным инструментом «лейка» в инвентаре, и нажатием определенной клавиши, сократит свой таймер смены статуса из «*grow*» в «*ready*» в результате чего будет расти быстрее. В случае если объект роста уже имеет статус «*ready*», инструмент не будет использован.

У инструментов в игре присутствуют эффекты. Их описывает класс *ToolEffect* (Приложение А, листинг кода *ToolEffect.cs*). Инструмент «корзинка» имеет совой эффект, его реализует класс *BasketToolEffect* (Приложение А, листинг кода *BasketToolEffect.cs*). Этот эффект срабатывает с определенным шансом при использовании собственно инструмента. При срабатывании эффекта, количество использований инструмента «лейка увеличивается», тем самым улучшая процесс выращивания. Эффект инструмента «лейка» реализует класс *WateringToolEffect* (Приложение А, листинг кода *WateringToolEffect.cs*), который с определенным шансом срабатывает при использовании собственно инструмента. При активации эффекта, скорость передвижения игрока увеличивается, также уменьшаются задержки на посадку и сбор урожая, в результате чего улучшая процесс выращивания культур.

Игровой процесс разделен на дни. Смену времени суток реализует класс *ChangingDay* (Приложение А, листинг кода *ChangingDay.cs*), унаследованный от *ScriptObject*. Содержит метод *Start*, где инициализируется время смены дня. Смена игрового дня знаменуется затемнением или осветлением заднего фона поля.

Сам класс фермы называется *Fermas* (Приложение А, листинг кода *Fermas.cs*), который наследуется от класса *Scene* (Приложение А, листинг кода *Scene.cs*) и содержит метод, который создает задний фон на игровом поле. Метод *СreateFermasElements*, позволяет при наличии указанного *RGB* цвета и метода *CreateObject*, *CreateGrowingTerritory* и т.д., создавать элементы фермы. Метод *RemoveObjectFromScene*, который удаляет объект из листа отрисовки и сам метод *EndScene*, который при наличии заданного в лаунчере количества монет (по умолчанию 500) у игрока, заканчивает игру. В случае, если игрок не успел набрать заданное количество монет за отведенное количество дней, игра также заканчивается.

При окончании игры закрывается окно формы *RenderForm* и в лаунчер выводится количество набранных монет и количество пройденных дней.

## **3.2 Тесты игрового приложения «ФЕРМА»**

Решение включает в себя проект под названием *TestGameApp*, который предназначен для тестирования разработанных библиотек игрового приложения.

На рисунке 3.1 представлен результат тестирования классов разработанного игрового приложения.

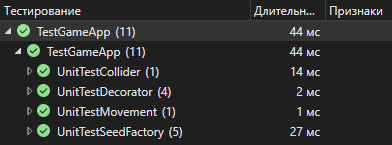


Рисунок 3.1 – Результаты тестирования классов разработанного игрового приложения

Классы, назначение и методы:

*– UnitTestCollider* (Приложение А, листинг кода *UnitTestCollider.cs*) – класс, предназначенный для тестирования разных видов столкновений, содержит метод *TestCollider*. Как пример рассмотрен случай, где создается два игровых объекта. Отталкиваясь от их расположения и тегов, определяется наличие пересечения;

*– UnitTestMovement* (Приложение А, листинг кода*UnitTestMovement.cs*) – класс, предназначенный для тестирования движения игрового объекта. Как пример рассмотрен случай, когда создается игровой объект, задается начальное положение, задается вектор смещения (*offset*), осуществляется перемещение объекта, сравнивается ожидаемый результат с текущим.

– *UnitTestDecorator* – класс, предназначенный для тестирования декораций объекта. Содержит методы:

– *DecoratedReed*;

– *DecoratedInfernalGrowth*;

– *DecoratedWheat*;

– *DecoratedSunFlower*.

Рассмотрен момент, когда один из классов подвергается декорации. Проверяется итоговое значение определенного поля.

**–** *UnitTestSeedFactory* – класс, предназначенный для тестирования фабрик создания растений. Содержит методы:

– *TestPumpkinFactory*;

– *TestWheatFactory*;

– *TestReedFactory*;

– *TestSunFlowerFactory*;

– *TestInfernalGrowthFactor*.

Рассмотрен момент создания игрового объекта типа растения. Проверяется наименования определенного экземпляра игрового объекта с ожидаемым.

## **3.3 Результат верификации приложения**

При запуске проекта открывается окно *Windows Form* c названием *Launch*. Над надписью «*player*» над окном скинов игрока находятся кнопки, при нажатии на которые буду показаны возможные состояния игрового персонажа. В правой части интерфейса имеется возможность изменения исходной настройки количества дней и монет, которых должен пройти и набрать игрок. Нажав на стрелки с обозначением справа, количество дней, за которые игрок должен набрать определенное количество монет увеличится. Нажав на стрелки с обозначение слева, количество уменьшится соответственно. На рисунке 3.2 показан интерфейс формы после запуска программы (до начала игры).



Рисунок 3.2 – Интерфейс игрового приложения

Нажав на левую кнопку, будет виден скин игрового персонажа в исходном состоянии, нажав на кнопку, которая находится по центру, можно увидеть скин игрового персонажа под эффектом ускорения, нажав на левую кнопку, можно увидеть состояния скина игрового персонажа в моменте, когда пополняется инструмент лейка. В случае, если ни одна кнопка не будет нажата, на иконке отображения скинов игрового персонажа будет отображаться пустое поле. На рисунке 3.3 показаны скины игрока после нажатия на кнопки эффектов.

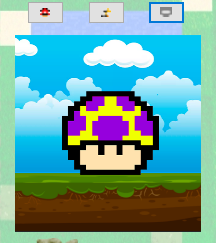
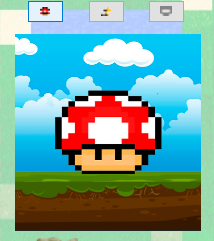
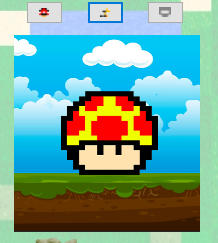


Рисунок 3.3 – Скины игрока в результате нажатия на кнопки эффектов

Игрок появляется на сцене в месте, заданном в хранимой структуре поля формата *.bmp*. Изначально, скорость игрока равна 4, но если при сборе урожая выпадет бонус, то скорость игрока увеличивается в 2 раза на определенное время.

На рисунке 3.5 приведена информация об управлении игровым персонажем.



Рисунок 3.5 – Информация об управлении игровым персонажем

Цель игрока – набрать определенное количество монет за отведенное количество дней. Монеты игрок зарабатывает, собирая урожай.

На рисунке 3.6 показаны культуры, с которыми может взаимодействовать игрок.



Рисунок 3.6 – Культуры, с которыми может взаимодействовать игрок

Карта разделена на секции, каждая из которых представляет из себя определенную территорию, на которой можно выращивать только определенную культуру. После того, как пользователь нажмет на кнопку c наименованием «*play*» начнется игра. В самом начале лучше будет посадить семена, территория, для посадки которых находится вблизи, иначе, можно потратить время впустую. Приоритет нужно отдавать растению «адский нарост», изображенному на рисунке 3.6 красным цветом, так как за сбор такого растения дается максимально доступное количество монет за сбор культуры.

На рисунке 3.7 показан пример рассадки культур по определенным секциям.

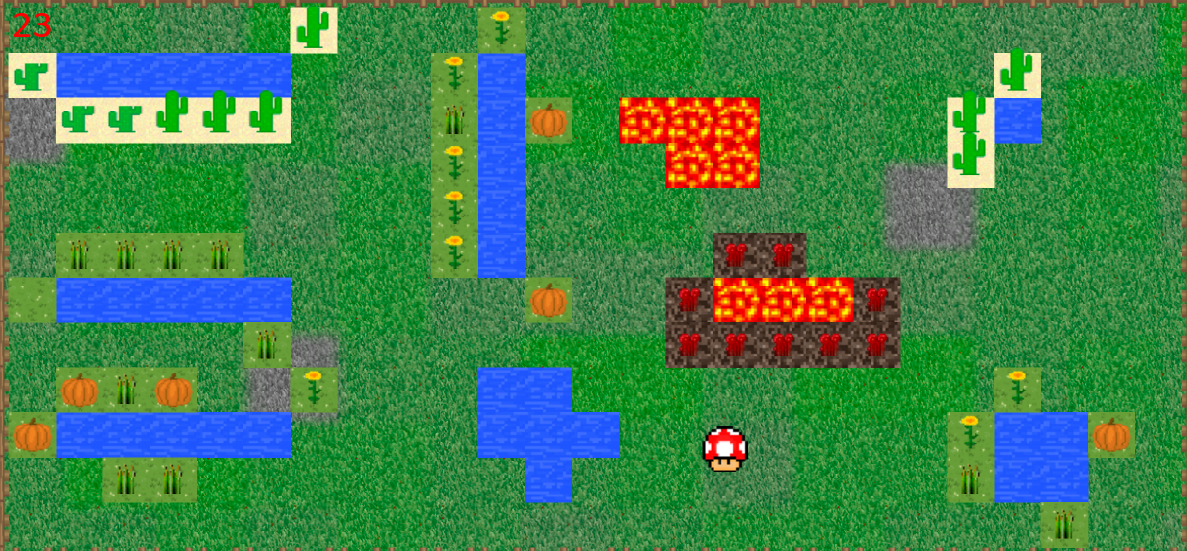


Рисунок 3.7 – Пример рассадки культур по определенным секциям

На рисунке 3.8 продемонстрирован пример отображения заработанных игроком монет.



Рисунок 3.8 – Пример отображения заработанных монет

Вначале игры игроку выдается случайное количество используемых ресурсов.

На рисунке 3.9 продемонстрирован инвентарь игрока с отображением его ресурсов.



Рисунок 3.9 – Отображение инвентаря и ресурсов игрока

Для того, чтобы двигать прогресс прохождения игроку придется использовать инструменты. Игрок, используя инструмент «корзина», может собирать урожай тем самым получать монеты. За сбор тыквы игрок получит 6 монет, за пшеницу 2 монеты, за подсолнух 5 монет, за кактус 4 монеты, за адский нарост 12 монет. Таким образом, при наличии семян адского нароста лучше всегда пытаться «пускать их в ход», но также стоит учитывать, что семена адского нароста растут дольше всего. Используя инструмент «лейка» на семена, растения будут расти быстрее.

На рисунке 3.10 показаны инструменты, с помощью которых игрок может взаимодействовать с культурами.



Рисунок 3.10 – Инструменты, с помощью которых игрок может взаимодействовать с культурами

Основным компонентом игры является сам уровень, который позволяет по каждому пикселю создавать элементы фермы. На рисунке 3.11 показана *.bmp* структура поля.

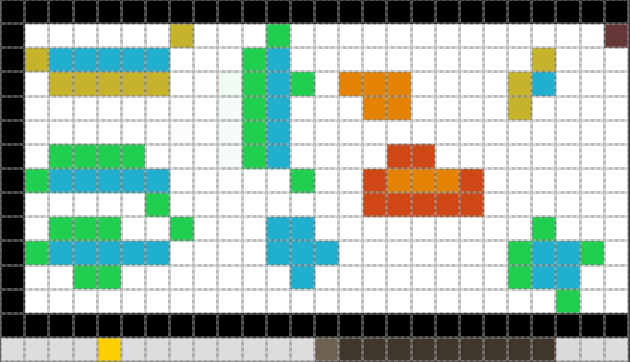


Рисунок 3.11 – Карта уровня

Каждый цвет на рисунке 3.11 соответствует своему элементу поля.

Элементы:

– забор (черный цвет);

– подсчитываемые ячейки (темно-коричневый);

– ячейка корзинки (светло-коричневый);

– монета (желтый);

– песок (темно-желтый);

– земля (зеленый);

– песок душ (темно-оранжевый);

– лава (светло-оранжевый);

– вода (синий);

– появление игрока (деревянный).

После окончания игры выводится окно *Windows Forms*, где выводится информация о достигнутой статистике. Информация о том, сколько монет набрал пользователь отобразится выше, а информация о пройденных днях отобразится ниже.

**ЗАКЛЮЧЕНИЕ**

В результате проделанной работы удалось разработать однопользовательское игровое приложение «ФЕРМА» с использованием графики *DirectX*. В игре игрок играет за одного персонажа и преодолевая любые трудности одержать победу.

Для программной реализации приложения использовался объектно-ориентированный язык *С#* с графической оболочкой *SharpDX*. Были изучены основные принципы работы с окном *DirectX*. Использование паттерна декоратора и фабричного метода для приложения позволило эффективно использовать его функционал.

В процессе разработки игрового приложения были рассмотрены:

– основные подходы к созданию игр;

– технологии *DirectX* и *OpenGL*.

С помощью декомпозиции была определена структура и алгоритм разработки программного обеспечения. Отсюда можно выделить 2 этапа разработки:

– этап разработки игрового движка;

– этап разработки логики игры.

Разработанные классы были протестированы юнит-тестами и, исходя из выявленных ошибок, отлажены, что гарантирует корректную работу приложения и делает ошибку, которую может получить пользователь в процессе выполнения, с минимальным шансом. Последним этапом разработки стало создание пользовательского интерфейса, с помощью которого можно запустить игру, задать некоторые исходные денные (если понадобится), отобразить статистику.

Игра имеет минимальный порог вхождения, интуитивно понятна и, как следствие, не требует особых навыков. Соревновательные элементы игры гарантируют полное погружение в игровой процесс, получение интересного опыта, оказание положительного влияния на такие спектры развития, как логическое мышление и внимательность, что в совокупности повышает значимость и ценность игры как программного продукта.

Курсовая работа проверена в системе «Антиплагиат». Процент оригинальности составляет 95,5 процента. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источниках».

Авторские права на программную часть проекта принадлежат автору курсовой работы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. GamesBeat [Электронный ресурс]. – 2022. – Режим доступа: https://venturebeat.com/2019/03/24/the-truth-about-hypercasual-games – Дата доступа: 20.05.2022;

2. Allen, S. Beginning DirectX 11: Course Technology PTR / A. Sherrod.– Австралия, Бразилия Япония, Корея, Мексика, Сингапур, Испания, Соединенные Штаты :Course Technology, a part of Cengage Learning,2011.-385;

3. Каверин, М. OpenGL Red Book / М. Каверин.–М.: 2010.-364;

4. Приемы объектно-ориентированного программирования. Паттерны проектирования / Э. Гамма – СПб.: Питер, 2015.-368;

5. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке С#. / Дж. Рихтер. – СПб: Питер, 2019.-896;

6. Приемы объектно-ориентированного программирования. Паттерны проектирования / Э. Гамма – СПб.: Питер, 2015.-111;

7. Microsoft Docs [Электронный ресурс]. – 2022. – Режим доступа: https://docs.microsoft.com/ru-ru/nuget/what-is-nuget – Дата доступа: 20.05.2022.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы «ФЕРМА»**

**Код программы для ApplicationRendering.cs**:

using System;

using SharpDX.Direct2D1;

using SharpDX.Windows;

namespace EngineLib.EngineComponents

{

public class ApplicationRendering : IDisposable

{

private readonly RenderForm \_renderForm;

private readonly WindowRenderTarget \_windowRenderTarget;

public static ApplicationRendering Instance { get; }

private InputHandler \_inputHandler { get; set; }

public InputHandler InputHandler { get { return \_inputHandler; } }

private Scene \_scene { get; set; }

public static float CurrentTime { get; set; }

private const float UNIT\_HEIGHT = 15.0f;

private const int WITHD\_APPLICATION = 1280;

private const int HEIGHT\_APPLICATION = 720;

public ApplicationRendering()

{

\_renderForm = new RenderForm("Fermas");

\_renderForm.TopMost = false;

\_renderForm.StartPosition = System.Windows.Forms.FormStartPosition.Manual;

\_renderForm.Height = HEIGHT\_APPLICATION;

\_renderForm.Width = WITHD\_APPLICATION;

\_renderForm.AllowUserResizing = true;

RenderSystem renderSystem = new RenderSystem(\_renderForm);

\_windowRenderTarget = renderSystem.WindowRenderTarget;

\_inputHandler = new InputHandler();

TextElement.SetRenderTarget(\_windowRenderTarget);

Input.InputHandlerInput(\_inputHandler);

}

public void SetScene(Scene scene)

{

\_scene = scene;

scene.SceneInit(\_windowRenderTarget, HEIGHT\_APPLICATION / UNIT\_HEIGHT);

}

public void Run()

{

RenderLoop.Run(\_renderForm, RenderCallback);

}

private void RenderCallback()

{

Time.UpdateTime();

\_inputHandler.UpdateKeyboardState();

\_windowRenderTarget.BeginDraw();

\_windowRenderTarget.Clear(SharpDX.Color.Black);

\_scene.DrawScene();

\_windowRenderTarget.EndDraw();

if (\_scene.\_isDrawedScnene)

{

Dispose();

\_scene.CloseForm();

}

}

public void Dispose()

{

\_renderForm.Dispose();

\_windowRenderTarget.Dispose();

\_scene.Dispose();

}

}

}

**Код программы для Input.cs:**

using SharpDX.DirectInput;

namespace EngineLib.EngineComponents

{

public enum MoveAxis

{

Horizontal,

Vertical,

Space

}

public enum InventoryChoice

{

One,

Two,

Three,

Four,

Five,

Six,

Seven,

}

public enum Reset

{

Esc,

}

public static class Input

{

public static InputHandler InputHandler { get; private set; }

public static void InputHandlerInput(InputHandler inputHandler)

{

InputHandler = inputHandler;

}

public static int GetAxisVector(MoveAxis moveAxis)

{

if (InputHandler == null)

return 0;

int movement = 0;

switch (moveAxis)

{

case MoveAxis.Horizontal:

{

if (InputHandler.KeyBoardState.IsPressed(Key.A)) movement--;

if (InputHandler.KeyBoardState.IsPressed(Key.D)) movement++;

break;

}

case MoveAxis.Vertical:

{

if (InputHandler.KeyBoardState.IsPressed(Key.W)) movement--;

if (InputHandler.KeyBoardState.IsPressed(Key.S)) movement++;

break;

}

}

return movement;

}

public static bool GatPozitionMove(MoveAxis axis)

{

if (InputHandler == null)

return false;

if (InputHandler.KeyBoardState.IsPressed(Key.Space)) return true;

else return false;

}

public static bool IsEsc(Reset reset)

{

if(InputHandler.KeyBoardState.IsPressed(Key.Escape)) return true;

return false;

}

public static bool GetInventoryChoice(InventoryChoice inventoryChoice)

{

bool choice = false;

if (InputHandler == null)

{

choice = false;

}

else

{

switch (inventoryChoice)

{

case InventoryChoice.One:

{

if (InputHandler.KeyBoardState.IsPressed(Key.D1)) choice = true;

break;

}

case InventoryChoice.Two:

{

if (InputHandler.KeyBoardState.IsPressed(Key.D2)) choice = true;

break;

}

case InventoryChoice.Three:

{

if (InputHandler.KeyBoardState.IsPressed(Key.D3)) choice = true;

break;

}

case InventoryChoice.Four:

{

if (InputHandler.KeyBoardState.IsPressed(Key.D4)) choice = true;

break;

}

case InventoryChoice.Five:

{

if (InputHandler.KeyBoardState.IsPressed(Key.D5)) choice = true;

break;

}

case InventoryChoice.Six:

{

if (InputHandler.KeyBoardState.IsPressed(Key.D6)) choice = true;

break;

}

case InventoryChoice.Seven:

{

if (InputHandler.KeyBoardState.IsPressed(Key.D7)) choice = true;

break;

}

default:

{

choice = false;

break;

}

}

}

return choice;

}

}

}

**Код программы для InputHandler.cs:**

using System;

using SharpDX;

using SharpDX.DirectInput;

namespace EngineLib.EngineComponents

{

public class InputHandler

{

private DirectInput \_directInput { get; }

private Keyboard keyboard;

private KeyboardState \_keyboardState;

public KeyboardState KeyBoardState

{

get { return \_keyboardState; }

set { \_keyboardState = value; }

}

public bool IsKeyboardAccess { get { return \_isKeyboardControlAccess; } set { \_isKeyboardControlAccess = value; } }

private bool \_isKeyboardControlAccess;

private bool \_isKeyboardUpdated;

public bool IsKeyboardUpdated

{

get { return \_isKeyboardUpdated; }

set { \_isKeyboardUpdated = value; }

}

public KeyboardState KeyborardState

{

get { return \_keyboardState; }

set { \_keyboardState = value; }

}

public InputHandler()

{

\_directInput = new DirectInput();

keyboard = new Keyboard(\_directInput);

keyboard.Properties.BufferSize = 20;

IsKeyboardAccess = IsKeyboardControlAccess();

\_keyboardState = new KeyboardState();

}

public bool IsKeyboardControlAccess()

{

try

{

keyboard.Acquire();

return true;

}catch (Exception ex)

{

return false;

}

}

public void UpdateKeyboardState()

{

if (!IsKeyboardAccess) IsKeyboardAccess = IsKeyboardControlAccess();

ResultDescriptor resultCode = ResultCode.Ok;

try

{

keyboard.GetCurrentState(ref \_keyboardState);

\_isKeyboardUpdated = true;

}

catch (SharpDXException e)

{

resultCode = e.Descriptor;

\_isKeyboardUpdated = false;

}

if (resultCode == ResultCode.InputLost || resultCode == ResultCode.NotAcquired)

IsKeyboardAccess = false;

}

}

}

**Код программы для RenderSystem.cs:**

using SharpDX;

using SharpDX.Direct2D1;

using SharpDX.WIC;

using Direct2D1 = SharpDX.Direct2D1;

using WIC = SharpDX.WIC;

using SharpDX.Windows;

using RenderTargetFactory = SharpDX.Direct2D1.Factory;

using ImageFactory = SharpDX.WIC.ImagingFactory;

namespace EngineLib.EngineComponents

{

public class RenderSystem

{

public WindowRenderTarget WindowRenderTarget { get; private set; }

private RenderTargetFactory \_renderTargetFactory { get; }

private static ImageFactory \_imageFactory { get; set; }

private static WindowRenderTarget \_windowRenderTarget { get; set; }

public RenderSystem(RenderForm Form)

{

\_renderTargetFactory = new RenderTargetFactory();

\_imageFactory = new ImageFactory();

RenderTargetProperties renderWindowTargetProperties = new RenderTargetProperties

{

DpiX = 0,

DpiY = 0,

MinLevel = FeatureLevel.Level\_10,

PixelFormat = new Direct2D1.PixelFormat(SharpDX.DXGI.Format.B8G8R8A8\_UNorm, AlphaMode.Premultiplied),

Type = RenderTargetType.Hardware,

Usage = RenderTargetUsage.None

};

HwndRenderTargetProperties hwdWindowTargetProperties = new HwndRenderTargetProperties

{

Hwnd = Form.Handle,

PixelSize = new Size2(Form.Width, Form.Height),

PresentOptions = PresentOptions.None

};

WindowRenderTarget = new WindowRenderTarget(\_renderTargetFactory, renderWindowTargetProperties, hwdWindowTargetProperties);

\_windowRenderTarget = WindowRenderTarget;

}

public static Direct2D1.Bitmap LoadBitmap(string imageFileName)

{

BitmapDecoder decoder = new BitmapDecoder(\_imageFactory, imageFileName, DecodeOptions.CacheOnDemand);

BitmapFrameDecode frame = decoder.GetFrame(0);

FormatConverter converter = new FormatConverter(\_imageFactory);

converter.Initialize(frame, WIC.PixelFormat.Format32bppPRGBA, BitmapDitherType.Ordered4x4, null, 0.0, BitmapPaletteType.Custom);

Direct2D1.Bitmap bitmap = Direct2D1.Bitmap.FromWicBitmap(\_windowRenderTarget, converter);

Utilities.Dispose(ref converter);

Utilities.Dispose(ref frame);

Utilities.Dispose(ref decoder);

return bitmap;

}

}

}

**Код программы для Scene.cs:**

using System.Collections.Generic;

using EngineLib.ObjectComponents;

using SharpDX.Direct2D1;

using SharpDX;

namespace EngineLib.EngineComponents

{

public abstract class Scene

{

public WindowRenderTarget \_windowRenderTarget { get; private set; }

private float \_scale { get; set; }

public List<GameObject> gameObjects { get; private set; }

protected List<GameObject> gameObjectsAdd { get; private set; }

protected List<GameObject> gameObjectsDrop { get; private set; }

public bool \_isDrawedScnene = false;

public List<Vector2> EmptyObjects { get; private set; }

public void SceneInit(WindowRenderTarget windowRenderTarget, float Scale )

{

\_windowRenderTarget = windowRenderTarget;

\_scale = Scale;

gameObjects = new List<GameObject>();

EmptyObjects = new List<Vector2>();

gameObjectsAdd = new List<GameObject>();

gameObjectsDrop = new List<GameObject>();

CreateScenesGameObjects();

}

public void SetObjectsComponents(List<GameObject> gameObjects)

{

foreach (GameObject gameObject in gameObjects)

{

gameObject.EngineComponentsInit(\_windowRenderTarget, \_scale);

}

}

public abstract void CreateScenesGameObjects();

public void DrawScene()

{

SetObjectsComponents(gameObjects);

if(\_isDrawedScnene)

{

gameObjects.Clear();

gameObjectsAdd.Clear();

gameObjectsDrop.Clear();

EmptyObjects.Clear();

return;

}

AddObjectsToScene();

RemoveObjectsFromScene();

foreach (GameObject gameObject in gameObjects)

{

gameObject.Draw();

}

}

public virtual void EndScene()

{

\_isDrawedScnene = true;

}

public abstract void CloseForm();

public void RemoveObjectsFromScene()

{

foreach (GameObject gameobject in gameObjectsDrop)

{

gameObjects.Remove(gameobject);

}

gameObjectsDrop.Clear();

}

public void AddObjectsToScene()

{

gameObjects.AddRange(gameObjectsAdd);

gameObjectsAdd.Clear();

}

public void Dispose()

{

\_windowRenderTarget = null;

}

}

}

**Код программы для Time.cs:**

using System;

using System.Diagnostics;

namespace EngineLib.EngineComponents

{

public class Time

{

private static Stopwatch watch;

private static long previousTicks;

/// <summary>

/// Текущее время с запуска приложения

/// </summary>

public static float CurrentTime { get; private set; }

/// <summary>

/// Разница во времени между кадрами

/// </summary>

public static float DeltaTime { get; private set; }

/// <summary>

/// Конструкторы статического класса

/// </summary>

static Time()

{

watch = new Stopwatch();

Reset();

}

/// <summary>

/// Обновление подсчитанных значений

/// </summary>

public static void UpdateTime()

{

long ticks = watch.Elapsed.Ticks;

CurrentTime = (float)ticks / TimeSpan.TicksPerSecond;

DeltaTime = (float)(ticks - previousTicks) / TimeSpan.TicksPerSecond;

previousTicks = ticks;

}

/// <summary>

/// Сброс таймера и счетчика

/// </summary>

public static void Reset()

{

watch.Reset();

watch.Start();

previousTicks = watch.Elapsed.Ticks;

}

}

}

Код программы для Collider.cs:

using System.Collections.Generic;

using System.Threading.Tasks;

using SharpDX;

namespace EngineLib.ObjectComponents

{

public class Collider

{

public GameObject gameObject { get; private set; }

private static List<GameObject> collidersOfGameObjects { get; set; }

private Vector2 ColliderPoint { get; set; }

public Size2F ColliderScale { get; private set; }

private Vector2[] boundCorners { get; set; }

private Vector2 offsetCollider { get; set; }

public bool IsInactive { get; private set; }

public Collider(GameObject gameObject, Size2F scale, Vector2 offset = new Vector2())

{

if (collidersOfGameObjects == null)

collidersOfGameObjects = new List<GameObject>();

collidersOfGameObjects.Add(gameObject);

this.gameObject = gameObject;

boundCorners = new Vector2[4];

ColliderScale = scale;

offsetCollider = offset;

IsInactive = false;

}

/// <summary>

/// Обновление границ твердого тела

/// </summary>

private void UpdateBounds()

{

Vector2 position = gameObject.Transform.Position;

float offsetWidth = ColliderScale.Width / 2;

float offsetHeight = ColliderScale.Height / 2;

boundCorners[0] = new Vector2(position.X + offsetCollider.X - offsetWidth, position.Y + offsetCollider.Y - offsetHeight);

boundCorners[1] = new Vector2(position.X + offsetCollider.X - offsetWidth, position.Y + offsetCollider.Y + offsetHeight);

boundCorners[2] = new Vector2(position.X + offsetCollider.X + offsetWidth, position.Y + offsetCollider.Y + offsetHeight);

boundCorners[3] = new Vector2(position.X + offsetCollider.X + offsetWidth, position.Y + offsetCollider.Y - offsetHeight);

}

/// <summary>

/// Проверка на пересечние компонента твердого тела с другими компонентами твердого тела, имеющие тег у игрового объекта

/// </summary>

/// <param name="tagNames">Теги игровых объектов, с которыми ожидается столкновение</param>

/// <returns>Реакция на проверку</returns>

public bool CheckIntersection(params string[] tagNames)

{

foreach (GameObject otherGameObject in collidersOfGameObjects)

{

if (otherGameObject == gameObject || otherGameObject.Collider.IsInactive) continue;

bool hasTag = false;

for (int i = 0; i < tagNames.Length && !hasTag; i++)

{

hasTag = otherGameObject.ObjectName == tagNames[i];

}

if (hasTag)

{

if (CheckGameObjectIntersection(otherGameObject))

{

return true;

}

}

}

return false;

}

public GameObject TagedGameObject(params string[] tagNames)

{

foreach (GameObject otherGameObject in collidersOfGameObjects)

{

if (otherGameObject == gameObject || otherGameObject.Collider.IsInactive) continue;

bool hasTag = false;

for (int i = 0; i < tagNames.Length && !hasTag; i++)

{

hasTag = otherGameObject.ObjectName == tagNames[i];

}

if (hasTag)

{

if (CheckGameObjectIntersection(otherGameObject))

{

return otherGameObject;

}

}

}

return null;

}

/// <summary>

/// Проверка на пересечние компонента твердого тела с другими компонентами твердого тела, имеющие тег у игрового объекта

/// </summary>

/// <param name="intersecredGameObject">Пересекаемый объект</param>

/// <param name="tagNames">Теги игровых объектов, с которыми ожидается столкновение</param>

/// <returns>Реакция на проверку</returns>

public bool CheckIntersection(out GameObject intersecredGameObject, params string[] tagNames)

{

foreach (GameObject otherGameObject in collidersOfGameObjects)

{

if (otherGameObject == gameObject || otherGameObject.Collider.IsInactive) continue;

bool hasTag = false;

for (int i = 0; i < tagNames.Length && !hasTag; i++)

{

hasTag = otherGameObject.ObjectName == tagNames[i];

}

if (hasTag)

{

if (CheckGameObjectIntersection(otherGameObject))

{

intersecredGameObject = otherGameObject;

return true;

}

}

}

intersecredGameObject = null;

return false;

}

/// <summary>

/// Проверка на пересечение компонента твердого тела с другими компонентами твердого тела, имеющие конкретный сценарий выполнения T

/// </summary>

/// <typeparam name="T">Конкретный сценарий выполнения</typeparam>

/// <param name="objectScript">Сценарий выполнения игрового объекта, с которым столкнулся</param>

/// <returns>Реакция на столкновение</returns>

public bool CheckIntersection<T>(out T objectScript)

where T : ScriptObject

{

foreach (GameObject otherGameObject in collidersOfGameObjects)

{

if (otherGameObject == gameObject || otherGameObject.scriptObject == null || otherGameObject.Collider.IsInactive) continue;

if (otherGameObject.scriptObject is T)

{

if (CheckGameObjectIntersection(otherGameObject))

{

objectScript = (T)otherGameObject.scriptObject;

return true;

}

}

}

objectScript = null;

return false;

}

/// <summary>

/// Проверка на пересечение компонента твердого тела с другим компонентам твердого тела

/// </summary>

/// <param name="otherGameObject">Игровой объект с компонетом твердого тела</param>

/// <returns>Реакция на столкновение</returns>

private bool CheckGameObjectIntersection(GameObject otherGameObject)

{

UpdateBounds();

otherGameObject.Collider.UpdateBounds();

Collider collider = otherGameObject.Collider;

int count = boundCorners.Length + collider.boundCorners.Length;

Vector2[] allCorners = new Vector2[count];

boundCorners.CopyTo(allCorners, 0);

collider.boundCorners.CopyTo(allCorners, boundCorners.Length);

Vector2 normal;

bool isInteresect = false;

for (int i = 0; i < count && !isInteresect; i++)

{

normal = GetNormal(allCorners, i);

Vector2 currentProjection = GetProjection(normal);

Vector2 otherProjection = collider.GetProjection(normal);

if (currentProjection.X < otherProjection.Y || otherProjection.X < currentProjection.Y)

{

return false;

}

}

return true;

}

/// <summary>

/// Создание нормали

/// </summary>

/// <param name="corners">Углы двух компонетов</param>

/// <param name="index">Номер угла</param>

/// <returns>Нормаль</returns>

private Vector2 GetNormal(Vector2[] corners, int index)

{

int next = index + 1;

next = next == corners.Length ? 0 : next;

Vector2 firstPoint = corners[index];

Vector2 secondPoint = corners[next];

Vector2 edge = new Vector2(secondPoint.X - firstPoint.X, secondPoint.Y - firstPoint.Y);

return new Vector2(-edge.Y, edge.X);

}

/// <summary>

/// Создание проекции

/// </summary>

/// <param name="normal">Нормаль</param>

/// <returns>Проекцию</returns>

private Vector2 GetProjection(Vector2 normal)

{

Vector2 result = new Vector2();

bool isNull = true;

foreach (Vector2 current in boundCorners)

{

float projection = normal.X \* current.X + normal.Y \* current.Y;

if (isNull)

{

result = new Vector2(projection, projection);

isNull = false;

}

if (projection > result.X)

result.X = projection;

if (projection < result.Y)

result.Y = projection;

}

return result;

}

}

}

**Код программы для GameObject.cs:**

using SharpDX;

using SharpDX.Direct2D1;

using EngineLib.EngineComponents;

namespace EngineLib.ObjectComponents

{

public class GameObject

{

public string ObjectName { get; private set; }

private WindowRenderTarget \_windowRenderTarget { get; set; }

private float \_worldScale { get; set; }

public Collider Collider { get; private set; }

public Transform Transform { get; private set; }

public Sprite Sprite { get; private set; }

public TextElement Text { get; private set; }

public ScriptObject scriptObject { get; private set; }

public bool IsCellObject { get; set; } = false;

public GameObject ParentGameobject { get; set; }

public GameObject ChildGameobject { get; set; }

public void EngineComponentsInit(WindowRenderTarget windowRenderTarget, float worldScale)

{

\_windowRenderTarget = windowRenderTarget;

\_worldScale = worldScale;

}

public void ObjectComponentsInit(string objectName, Transform transform, Sprite sprite = null, Collider collider = null/\*, Text text = null\*/)

{

ObjectName = objectName;

Transform = transform;

Sprite = sprite;

Collider = collider;

}

public void TextInit(TextElement text)

{

Text = text;

;

this.IsCellObject = true;

}

//public void SetParentObject(GameObject gameObject)

//{

// ParentGameobject = gameObject;

//}

public void SetTextString(string str)

{

this.Text.SetText(str);

}

public void ScriptInit(ScriptObject scriptObject)

{

this.scriptObject = scriptObject;

this.scriptObject.Initialize(this);

this.scriptObject.Start();

}

public void UpdateObjectSprite(GameObject gameObject, Sprite sprite)

{

Sprite = sprite;

}

public void Draw()

{

scriptObject?.Update();

if (Sprite == null) return;

Vector2 translation = new Vector2(0, 0);

translation.X += Transform.Position.X \* Transform.Scale.Width;

translation.Y += Transform.Position.Y \* Transform.Scale.Height;

Matrix3x2 matrix = Matrix3x2.Rotation(0, translation);

matrix \*= Matrix3x2.Scaling(Transform.Scale.Width \* \_worldScale / Sprite.SpriteWidth, Transform.Scale.Height \* \_worldScale / Sprite.SpriteHeight, translation);

matrix \*= Matrix3x2.Translation(translation \* \_worldScale);

if (\_windowRenderTarget != null)

{

\_windowRenderTarget.Transform = matrix;

if (Sprite.Bitmap != null)

{

\_windowRenderTarget.DrawBitmap(Sprite.Bitmap, 1f, BitmapInterpolationMode.Linear);

}

if (Text != null && this.IsCellObject && Text.Textstring != null)

{

\_windowRenderTarget.DrawText(Text.Textstring, Text.\_textFormat, Text.\_rectF, Text.\_Brush);

}

}

}

}

}

**Код программы для ScriptObject.cs:**

namespace EngineLib.ObjectComponents

{

public abstract class ScriptObject

{

protected GameObject gameObject;

public void Initialize(GameObject gameObject)

{

this.gameObject = gameObject;

}

public abstract void Start();

/// <summary>

/// Обновление игрового объекта

/// </summary>

public abstract void Update();

}

}

**Код программы для Sprite.cs:**

using SharpDX.Direct2D1;

namespace EngineLib.ObjectComponents

{

public class Sprite

{

public Bitmap Bitmap { get; private set; }

public float SpriteWidth { get; private set; }

public float SpriteHeight { get; private set; }

public Sprite(Bitmap sprite)

{

Bitmap = sprite;

SpriteWidth = sprite.Size.Width;

SpriteHeight = sprite.Size.Height;

}

}

}

**Код программы для TextElement.cs:**

using SharpDX.Direct2D1;

using SharpDX;

using SharpDX.DirectWrite;

using TextFactory = SharpDX.DirectWrite.Factory;

namespace EngineLib.EngineComponents

{

public class TextElement

{

private TextFactory \_textFactory { get; set; }

public Vector2 Position { get; set; }

public float TextSize { get; set; }

public string Textstring { get; private set; }

private static WindowRenderTarget \_windowRenderTarget { get; set; }

public Brush \_Brush { get; private set; }

public TextFormat \_textFormat { get; private set; }

public RectangleF \_rectF { get; private set; }

public TextElement(Vector2 position, string FontFamily, float FontSize, Color color)

{

\_textFactory = new TextFactory();

Size2F clientSize = \_windowRenderTarget.Size;

\_rectF = new RectangleF()

{

Width = clientSize.Width,

Height = clientSize.Height,

Location = position,

};

\_textFormat = new TextFormat(\_textFactory, FontFamily, FontSize);

\_textFormat.ParagraphAlignment = ParagraphAlignment.Near;

\_textFormat.TextAlignment = TextAlignment.Leading;

\_Brush = new SolidColorBrush(\_windowRenderTarget, color);

}

public void SetText(string stringText)

{

this.Textstring = stringText;

}

public static void SetRenderTarget(WindowRenderTarget windowRenderTarget)

{

\_windowRenderTarget = windowRenderTarget;

}

}

}

**Код программы для Transform.cs:**

using SharpDX;

namespace EngineLib.ObjectComponents

{

public class Transform

{

private Vector2 \_position;

public Vector2 Position

{

get { return \_position; }

set { \_position = value; }

}

private Vector2 movementinCurrentFrame;

public Size2F Scale { get { return \_scale; } set { \_scale = value; } }

private Size2F \_scale;

public Transform(Vector2 pozition, Size2F scale = new Size2F())

{

Scale = scale;

\_position = pozition;

}

public void SetMovement(Vector2 movement)

{

movementinCurrentFrame = movement;

\_position.X += movement.X;

\_position.Y += movement.Y;

}

public void ResetMovment()

{

\_position.X -= movementinCurrentFrame.X;

\_position.Y -= movementinCurrentFrame.Y;

}

}

}

**Код программы для Fermas.cs:**

using EngineLib.EngineComponents;

using EngineLib.ObjectComponents;

using GameLib.Game;

using System.Drawing;

using SharpDX;

namespace GameLib.Ferm

{

public class Fermas : Scene

{

public static Fermas Instance { get; set; }

private const string \_COLOR\_MAP = @"Ferm\Sprites\Fermmap\Fermas.bmp";

private PlayerConstructor \_playerConstructor { get; set; }

private static Bitmap \_fermasObjectsMap { get; set; }

private ObjectsFactory \_objectsFactory { get; set; }

private InventoryObjectsFactory \_inventoryObjectsFactory { get; set; }

private int \_gameDays { get; set; }

private int \_coinsToWin { get; set; }

public int CurrentDays { get; set; }

public int CurrentMoney { get; set; }

public Fermas() : this(30, 500)

{

}

public Fermas(int GameDays, int Coins)

{

this.\_gameDays = GameDays;

this.\_coinsToWin = Coins;

}

public int GetGameDays()

{

return \_gameDays;

}

public int GetCoinsToWin()

{

return \_coinsToWin;

}

public override void CreateScenesGameObjects()

{

if (Instance == null)

Instance = this;

\_objectsFactory = new ObjectsFactory();

\_inventoryObjectsFactory = new InventoryObjectsFactory();

\_fermasObjectsMap = new Bitmap(\_COLOR\_MAP);

GameObject EmptyGameObject = null;

GameObject backgroundObject = \_objectsFactory.CreateBackGround(EmptyGameObject, new Vector2(0, 0), "Фонday");

gameObjects.Add(backgroundObject);

\_playerConstructor = new PlayerConstructor();

CreateFermasElements();

GameObject Player = \_playerConstructor.CreatePlayer();

CreateFermasGrowingTerritory();

CreateInventoryInterface("Coin", "Basket", "Watering", "Pumpkin", "Reed", "InfernalGrowth", "SunFlower", "Wheat");

gameObjects.Add(Player);

\_playerConstructor.CreateTools();

}

private void CreateFermasElements()

{

for (int i = 0; i < \_fermasObjectsMap.Height; i++)

{

for (int j = 0; j < \_fermasObjectsMap.Width; j++)

{

GameObject GameObject = null;

System.Drawing.Color color = \_fermasObjectsMap.GetPixel(j, i);

if (color.R == 0 && color.G == 0 && color.B == 0)

{

GameObject = \_objectsFactory.CreateObject(GameObject, new Vector2(j, i), "Wall");

}

else if (color.R == 33 && color.G == 175 && color.B == 207)

{

GameObject = \_objectsFactory.CreateObject(GameObject, new Vector2(j, i), "Water");

}

else if (color.R == 227 && color.G == 130 && color.B == 5)

{

GameObject = \_objectsFactory.CreateObject(GameObject, new Vector2(j, i), "Lava");

}

else if (color.R == 238 && color.G == 9 && color.B == 41)

{

GameObject = \_objectsFactory.CreateObject(GameObject, new Vector2(j, i), "SpawnToolsBlock");

}

else if (color.R == 102 && color.G == 55 && color.B == 55)

{

\_playerConstructor.PlayerStartPosition = new Vector2(j, i);

}

else

EmptyObjects.Add(new Vector2(j, i));

if (GameObject != null)

gameObjects.Add(GameObject);

}

}

}

private void CreateFermasGrowingTerritory()

{

for (int i = 0; i < \_fermasObjectsMap.Height; i++)

{

for (int j = 0; j < \_fermasObjectsMap.Width; j++)

{

GameObject GameObject = null;

System.Drawing.Color color = \_fermasObjectsMap.GetPixel(j, i);

if (color.R == 199 && color.G == 178 && color.B == 43)

{

GameObject = \_objectsFactory.CreateGrowingTerritory(GameObject, new Vector2(j, i), "Send");

}

else if (color.R == 33 && color.G == 207 && color.B == 80)

{

GameObject = \_objectsFactory.CreateGrowingTerritory(GameObject, new Vector2(j, i), "Land");

}

else if (color.R == 207 && color.G == 71 && color.B == 20)

{

GameObject = \_objectsFactory.CreateGrowingTerritory(GameObject, new Vector2(j, i), "ShowerSand");

}

else

EmptyObjects.Add(new Vector2(j, i));

if (GameObject != null)

gameObjects.Add(GameObject);

}

}

}

public void CreateInventoryInterface(params string[] CellsTags)

{

int CellIndex = 0;

if (CellsTags != null)

{

for (int i = 0; i < \_fermasObjectsMap.Height; i++)

{

for (int j = 0; j < \_fermasObjectsMap.Width; j++)

{

GameObject GameObject = null;

System.Drawing.Color color = \_fermasObjectsMap.GetPixel(j, i);

if (color.R == 253 && color.G == 207 && color.B == 0)

{

CellIndex++;

GameObject = \_inventoryObjectsFactory.CreateCoinCellObject(GameObject, new Vector2(j, i), CellsTags[CellIndex - 1], new Vector2(0.1f, 0.1f));

}

if (color.R == 110 && color.G == 96 && color.B == 82)

{

CellIndex++;

GameObject = \_inventoryObjectsFactory.CreateToolCellObject(GameObject, new Vector2(j, i), CellsTags[CellIndex - 1], new Vector2(0.1f, 0.1f));

}

if (color.R == 65 && color.G == 54 && color.B == 43)

{

if (CellIndex == CellsTags.Length)

break;

CellIndex++;

GameObject = \_inventoryObjectsFactory.CreateCellCounterObject(GameObject, new Vector2(j, i), CellsTags[CellIndex - 1], new Vector2(0.1f, 0.1f));

}

if(GameObject != null)

gameObjects.Add(GameObject);

}

}

}

}

public void AddObjectOnScene(GameObject gameObject)

{

gameObjectsAdd.Add(gameObject);

}

public void RemoveObjectOnScene(GameObject gameObject)

{

gameObjectsDrop.Add(gameObject);

}

public override void EndScene()

{

base.EndScene();

}

public override void CloseForm()

{

this.Dispose();

Instance = null;

GameIvents.EndGame.Invoke(CurrentDays, CurrentMoney);

}

}

}

**Код программы для ChangingDay.cs:**

using EngineLib.ObjectComponents;

using EngineLib.EngineComponents;

using GameLib.Ferm.Delay;

using GameLib.Game;

namespace GameLib.Ferm

{

public enum DayNight

{

Day,

Night

}

public class ChangingDay : ScriptObject

{

private Fermas \_fermasInstance { get; set; }

private static readonly float \_daynightDuration;

private Sprite \_daySprite;

private Sprite \_nightSprite;

private int \_currentdays { get; set; }

private static DelayTime \_delayTime { get; set; }

static public DayNight DayNight { get; set; }

static ChangingDay()

{

\_delayTime = new DelayTime(5.0f);

\_daynightDuration = \_delayTime.Delay \* 2;

}

public override void Start()

{

\_fermasInstance = Fermas.Instance;

\_daySprite = new Sprite(RenderSystem.LoadBitmap(@"Ferm\Sprites\BackGround\Фонday.png"));

\_nightSprite = new Sprite(RenderSystem.LoadBitmap(@"Ferm\Sprites\BackGround\Фонnight.png"));

}

public override void Update()

{

\_delayTime.DeleyUpdate();

if (\_delayTime.Opportunity())

{

gameObject.UpdateObjectSprite(gameObject, \_nightSprite);

}

else

{

gameObject.UpdateObjectSprite(gameObject, \_daySprite);

}

if (\_delayTime.CurrentTime >= \_daynightDuration + 0.2f)

{

\_delayTime.DelayReset();

\_currentdays++;

Fermas.Instance.CurrentDays = \_currentdays;

gameObject.SetTextString(\_currentdays.ToString());

}

if (\_currentdays == Fermas.Instance.GetGameDays())

{

Fermas.Instance.CurrentMoney = Player.staticPlayerInstatnce.Coins;

Fermas.Instance.EndScene();

}

}

}

}

**Код программы для ObjectsFactory:**

using EngineLib.ObjectComponents;

using Transform = EngineLib.ObjectComponents.Transform;

using EngineLib.EngineComponents;

using GameLib.Ferm.GrowScripts;

using SharpDX;

namespace GameLib.Ferm

{

public class ObjectsFactory

{

public GameObject CreateObject(GameObject gameObject, Vector2 position, string ObjectName)

{

gameObject = new GameObject();

gameObject.ObjectComponentsInit(ObjectName, new Transform(position, new Size2F(1f, 1f)),

new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Elements\{ObjectName}.png")),

new Collider(gameObject, new Size2F(0.7f, 0.7f), new Vector2(-0.15f, 0.45f)));

return gameObject;

}

public GameObject CreateBackGround(GameObject gameObject, Vector2 position, string ObjectName)

{

gameObject = new GameObject();

gameObject.ObjectComponentsInit(ObjectName, new Transform(position, new Size2F(27, 15)),

new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\BackGround\{ObjectName}.png")));

gameObject.TextInit(new TextElement(gameObject.Transform.Position + new Vector2(50.0f, 40.0f), "Calibri", 40.0f, SharpDX.Color.Red));

gameObject.ScriptInit(new ChangingDay());

return gameObject;

}

public GameObject CreateGrowingTerritory(GameObject gameObject, Vector2 position, string ObjectName)

{

gameObject = new GameObject();

gameObject.ObjectComponentsInit(ObjectName, new Transform(position, new Size2F(1f, 1f)),

new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\GrowTerritory\{ObjectName}.png")),

new Collider(gameObject, new Size2F(0.7f, 0.7f), new Vector2(-0.15f, 0.45f)));

switch(ObjectName)

{

case "Land":

{

gameObject.ScriptInit(new LandScript());

break;

}

case "Send":

{

gameObject.ScriptInit(new SendScript());

break;

}

case "ShowerSand":

{

gameObject.ScriptInit(new ShowerSandScript());

break;

}

}

return gameObject;

}

}

}

**Код программы для InventoryObjectsFactory.cs:**

using EngineLib.ObjectComponents;

using EngineLib.EngineComponents;

using SharpDX;

using GameLib.Ferm.Cells;

namespace GameLib.Ferm

{

public class InventoryObjectsFactory

{

public GameObject CreateCellCounterObject(GameObject gameObject, Vector2 pozition, string CellName, Vector2 offset = new Vector2())

{

gameObject = new GameObject();

pozition = GetOffsetVector(pozition, offset);

gameObject.ObjectComponentsInit(CellName, new Transform(pozition,new Size2F(1f, 1f)),

new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Cells\{CellName}.png")));

gameObject.TextInit(new TextElement(pozition, "Calibri", 15f, Color.Chartreuse));

switch(CellName)

{

case "Pumpkin":

{

gameObject.ScriptInit(new PumpkinCell());

break;

}

case "Reed":

{

gameObject.ScriptInit(new ReedCell());

break;

}

case "InfernalGrowth":

{

gameObject.ScriptInit(new InfernalGrowthCell());

break;

}

case "Wheat":

{

gameObject.ScriptInit(new WheatCell());

break;

}

case "SunFlower":

{

gameObject.ScriptInit(new SunFlowerCell());

break;

}

case "Watering":

{

gameObject.ScriptInit(new WateringCell());

break;

}

case "Coin":

{

gameObject.ScriptInit(new MoneyCell());

break;

}

}

return gameObject;

}

public GameObject CreateToolCellObject(GameObject gameObject, Vector2 pozition, string CellName, Vector2 offset = new Vector2())

{

gameObject = new GameObject();

pozition = GetOffsetVector(pozition, offset);

gameObject.ObjectComponentsInit(CellName, new Transform(pozition, new Size2F(1f, 1f)),

new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Cells\{CellName}.png")));

if (CellName == "Basket")

gameObject.ScriptInit(new BusketCell());

return gameObject;

}

public GameObject CreateCoinCellObject(GameObject gameObject, Vector2 pozition, string CellName, Vector2 offset = new Vector2())

{

gameObject = new GameObject();

pozition = GetOffsetVector(pozition, offset);

gameObject.ObjectComponentsInit(CellName, new Transform(pozition, new Size2F(1f, 1f)),

new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Cells\{CellName}.png")));

gameObject.TextInit(new TextElement(pozition + new Vector2(3f, -4f), "Calibri", 12f, Color.Cornsilk));

gameObject.ScriptInit(new MoneyCell());

return gameObject;

}

protected Vector2 GetOffsetVector(Vector2 position, Vector2 offset)

{

if (offset != null)

return position + offset;

else return position;

}

}

}

**Код программы для GameEvents.cs:**

namespace GameLib.Game

{

public static class GameIvents

{

public delegate void EndScene(int CurrentDays, int CurrentCoins);

public static EndScene EndGame;

}

}

**Код программы для Inventory.cs:**

using System;

using System.Collections.Generic;

using EngineLib.ObjectComponents;

namespace GameLib.Game

{

public class Inventory

{

private Dictionary<string, List<GameObject>> \_componentsObjectsDictionary { get; set; }

private Dictionary<string, int> \_componentsCountDictionary { get; set; }

private Dictionary<string, GameObject> \_toolsObjectsDictionary { get; set; }

private int \_inventorySize { get; set; }

private Inventory \_inventoryInstance { get; set; }

private Random \_random { get; set; }

private List<string> \_components { get; set; }

public void CreateInvetnroty(int InventorySize)

{

\_inventorySize = InventorySize;

\_random = new Random();

\_componentsObjectsDictionary = new Dictionary<string, List<GameObject>>();

\_componentsCountDictionary = new Dictionary<string, int>();

\_toolsObjectsDictionary = new Dictionary<string, GameObject>();

\_components = new List<string>();

\_components = InvetnroryComponentsInit(\_components);

\_componentsObjectsDictionary = CropsObjectsDictionaryInit(\_componentsObjectsDictionary);

\_componentsCountDictionary = CropsCountDictionaryInit(\_componentsCountDictionary);

if(\_inventoryInstance == null)

\_inventoryInstance = this;

}

private Dictionary<string, List<GameObject>> CropsObjectsDictionaryInit(Dictionary<string, List<GameObject>> cropsObjectsDictionary)

{

for (int i = 0; i < \_inventorySize; i++)

{

cropsObjectsDictionary.Add(\_components[i], new List<GameObject>());

}

return cropsObjectsDictionary;

}

private Dictionary<string, int> CropsCountDictionaryInit(Dictionary<string, int> cropsCountDictionary)

{

for (int i = 0; i < \_inventorySize; i++)

{

cropsCountDictionary.Add(\_components[i], \_random.Next(2,4));

}

return cropsCountDictionary;

}

public void ToolsObjectsDictionaryInit(string ToolName, GameObject toolObject)

{

if (!\_toolsObjectsDictionary.ContainsKey(ToolName))

\_toolsObjectsDictionary.Add(ToolName, toolObject);

}

public GameObject GetToolObject(string ToolName)

{

return \_toolsObjectsDictionary[ToolName];

}

public int GetComponentsCount(string CropName)

{

int value = 0;

if (\_componentsCountDictionary.TryGetValue(CropName, out value))

return value;

else return 0;

}

private List<string> InvetnroryComponentsInit(List<string> components)

{

components.Add("Watering");

components.Add("Pumpkin");

components.Add("Reed");

components.Add("InfernalGrowth");

components.Add("Wheat");

components.Add("SunFlower");

return components;

}

public void AddCmponentsCount(string objectName)

{

\_componentsCountDictionary[objectName]--;

}

public void RemoveComponentsCount(string objectName, int CountToAdd)

{

\_componentsCountDictionary[objectName] += CountToAdd;

}

public void AddCmponentsObject(GameObject gameObject)

{

\_componentsCountDictionary[gameObject.ObjectName]--;

List<GameObject> gameObjects;

\_componentsObjectsDictionary.TryGetValue(gameObject.ObjectName, out gameObjects);

if(gameObject != null)

\_componentsObjectsDictionary[gameObject.ObjectName].Add(gameObject);

}

public void RemoveComponentsObject(GameObject gameObject,int CountToAdd)

{

\_componentsCountDictionary[gameObject.ObjectName] += CountToAdd;

List<GameObject> gameObjects;

\_componentsObjectsDictionary.TryGetValue(gameObject.ObjectName, out gameObjects);

if (gameObject != null)

\_componentsObjectsDictionary[gameObject.ObjectName].Remove(gameObject);

}

public Inventory GetInventoryInstance()

{

return \_inventoryInstance;

}

}

}

**Код программы для Player.cs:**

using EngineLib.ObjectComponents;

using EngineLib.EngineComponents;

using GameLib.Ferm;

using SharpDX;

namespace GameLib.Game

{

public class Player : ScriptObject

{

private PlayerControl \_control { get; set; }

private const int \_TALLY\_OBJECTS = 6;

private Player \_playerInstance { get; set; }

public static Player staticPlayerInstatnce { get; private set; }

public static string choiseCell { get; set; }

public bool isSpeedEffected { get; private set; } = false;

public bool isAddEffected { get; private set; } = false;

private static Fermas \_fermasInstance { get; set; }

private Inventory \_inventory { get; set; }

public int Coins { get; private set; } = 0;

private Sprite \_effectedSpeedSprite { get; set; }

private Sprite \_effectedAddSprite { get; set; }

private Sprite \_uneffectedSprite { get; set; }

public static Inventory staticInventoryInstance { get; private set; }

private int \_speed { get; set; } = 4;

public override void Start()

{

staticPlayerInstatnce = \_playerInstance;

\_fermasInstance = Fermas.Instance;

\_control = new PlayerControl(MoveAxis.Horizontal, MoveAxis.Vertical, MoveAxis.Space,

InventoryChoice.One, InventoryChoice.Two, InventoryChoice.Three, InventoryChoice.Four, InventoryChoice.Five, InventoryChoice.Six, InventoryChoice.Seven);

\_inventory = new Inventory();

\_inventory.CreateInvetnroty(\_TALLY\_OBJECTS);

staticInventoryInstance = \_inventory.GetInventoryInstance();

\_effectedSpeedSprite = new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Player\EffectedPlayer\SpeedEffectPlayer.png"));

\_effectedAddSprite = new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Player\EffectedPlayer\AddEffectPlayer.png"));

\_uneffectedSprite = new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Player\Player.png"));

}

public void PlayerInit()

{

if (\_playerInstance == null)

\_playerInstance = this;

}

public Player GetPlayerInstance()

{

return \_playerInstance;

}

public override void Update()

{

if(Coins >= Fermas.Instance.GetCoinsToWin())

{

Fermas.Instance.CurrentMoney = Coins;

Fermas.Instance.EndScene();

}

if (isSpeedEffected)

gameObject.UpdateObjectSprite(gameObject, \_effectedSpeedSprite);

else if (isAddEffected)

gameObject.UpdateObjectSprite(gameObject, \_effectedAddSprite);

else

gameObject.UpdateObjectSprite(gameObject, \_uneffectedSprite);

PlayerMove();

PlayerInventoryChoise();

}

public void PlayerMove()

{

float NewXPosition = 0, NewYPosition = 0;

Vector2 direction;

Vector2 position;

NewXPosition = Input.GetAxisVector(\_control.Horizontal);

NewYPosition = Input.GetAxisVector(\_control.Vertical);

if(NewXPosition == 0)

{

direction = new Vector2(0, NewYPosition);

}

else

{

direction = new Vector2(NewXPosition, 0);

}

position = direction \* \_speed \* Time.DeltaTime;

gameObject.Transform.SetMovement(position);

DirectMoveCollision();

}

public void PlayerInventoryChoise()

{

if (Input.GetInventoryChoice(\_control.Three))

Player.choiseCell = "Pumpkin";

else if (Input.GetInventoryChoice(\_control.Four))

Player.choiseCell = "Reed";

else if (Input.GetInventoryChoice(\_control.Five))

Player.choiseCell = "InfernalGrowth";

else if (Input.GetInventoryChoice(\_control.Seven))

Player.choiseCell = "Wheat";

else if (Input.GetInventoryChoice(\_control.Six))

Player.choiseCell = "SunFlower";

else if (Input.GetInventoryChoice(\_control.One))

Player.choiseCell = "Basket";

else if (Input.GetInventoryChoice(\_control.Two))

Player.choiseCell = "Watering";

else if (Input.IsEsc(Reset.Esc))

Coins = 0;

}

public void DirectMoveCollision()

{

if (gameObject.Collider.CheckIntersection("Lava", "Wall", "Water"))

{

gameObject.Transform.ResetMovment();

}

}

public PlayerControl GetControl()

{

return \_control;

}

public void SetSpeed(int speed)

{

if(speed > 0)

{

\_speed = speed;

}

}

public void SetSpeedEffect(bool isEffect)

{

isSpeedEffected = isEffect;

}

public void SetAddEffect(bool isEffect)

{

isAddEffected = isEffect;

}

public void ChangeCoinsValue(int value)

{

Coins += value;

}

public int GetCoinsCount()

{

return Coins;

}

}

public struct PlayerControl

{

public MoveAxis Horizontal;

public MoveAxis Vertical;

public MoveAxis Space;

public InventoryChoice One;

public InventoryChoice Two;

public InventoryChoice Three;

public InventoryChoice Four;

public InventoryChoice Five;

public InventoryChoice Six;

public InventoryChoice Seven;

public PlayerControl(MoveAxis axis1, MoveAxis axis2, MoveAxis axis3, InventoryChoice one, InventoryChoice two, InventoryChoice three, InventoryChoice four, InventoryChoice five,

InventoryChoice six, InventoryChoice seven)

{

Horizontal = axis1;

Vertical = axis2;

Space = axis3;

One = one;

Two = two;

Three = three;

Four = four;

Five = five;

Six = six;

Seven = seven;

}

}

}

**Код программы для PlayerConstructor.cs:**

using EngineLib.ObjectComponents;

using SharpDX;

using EngineLib.EngineComponents;

using GameLib.Ferm.Tools.ToolsFactory;

using GameLib.Ferm.Tools;

using GameLib.Ferm;

using GameLib.Ferm.ToolEffects.ToolsEffectsFactory;

using GameLib.Ferm.ToolEffects;

namespace GameLib.Game

{

public class PlayerConstructor

{

public GameObject playerObject { get; private set; }

public Vector2 PlayerStartPosition { get; set; }

public GameObject CreatePlayer(string playerName = "Player")

{

GameObject gameObject = new GameObject();

gameObject.ObjectComponentsInit(playerName, new Transform(PlayerStartPosition, new Size2F(1f, 1f)), new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Player\{playerName}.png")),

new Collider(gameObject, new Size2F(0.5f, 0.5f), new Vector2(0,0.8f)));

playerObject = gameObject;

Player ScriptOfPlayer = new Player();

ScriptOfPlayer.PlayerInit();

gameObject.ScriptInit(ScriptOfPlayer);

return gameObject;

}

public void CreateTools()

{

ToolFactory toolFactory = new ToolFactory();

ToolEffectFactory toolEffectFactory = new ToolEffectFactory();

GameObject toolBasketObject = toolFactory.CreateTool(playerObject.Transform.Position, "Basket", new BasketTool());

toolBasketObject.ParentGameobject = playerObject;

Fermas.Instance.AddObjectOnScene(toolBasketObject);

Fermas.Instance.AddObjectOnScene(toolEffectFactory.CreateToolEffect(toolBasketObject, new BasketToolEffect()));

GameObject toolWateringObject = toolFactory.CreateTool(playerObject.Transform.Position, "Watering", new WateringTool());

toolWateringObject.ParentGameobject = playerObject;

Fermas.Instance.AddObjectOnScene(toolWateringObject);

Fermas.Instance.AddObjectOnScene(toolEffectFactory.CreateToolEffect(toolWateringObject, new WateringToolEffect()));

Player.staticInventoryInstance.ToolsObjectsDictionaryInit(toolBasketObject.ObjectName, toolBasketObject);

Player.staticInventoryInstance.ToolsObjectsDictionaryInit(toolWateringObject.ObjectName, toolWateringObject);

}

}

}

**Код программы для CellScript.cs:**

using EngineLib.ObjectComponents;

using GameLib.Game;

using EngineLib.EngineComponents;

namespace GameLib.Ferm.Cells

{

public abstract class CellScript : ScriptObject

{

private Inventory inventoryInstance { get; set; }

protected string playerChoise { get; private set; }

protected Sprite selectedCell { get; private set; }

protected Sprite unselectedCell { get; private set; }

public override void Start()

{

inventoryInstance = Player.staticInventoryInstance;

if (gameObject.ObjectName != "Coin")

selectedCell = new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Cells\ChoiseCells\{gameObject.ObjectName}.png"));

unselectedCell = new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Cells\{gameObject.ObjectName}.png"));

}

public override void Update()

{

playerChoise = Player.choiseCell;

if(gameObject.ObjectName == "Coin")

SeedCountInteraction(Player.staticPlayerInstatnce.GetCoinsCount());

else

SeedCountInteraction(inventoryInstance.GetComponentsCount(gameObject.ObjectName));

ChangeCellSprite();

}

public abstract void SeedCountInteraction(int Count);

public abstract void ChangeCellSprite();

}

}

**Код программы для BasketCell.cs:**

namespace GameLib.Ferm.Cells

{

public class BusketCell : CellScript

{

public override void ChangeCellSprite()

{

if(playerChoise == gameObject.ObjectName)

{

gameObject.UpdateObjectSprite(gameObject, selectedCell);

}

else gameObject.UpdateObjectSprite(gameObject, unselectedCell);

}

public override void SeedCountInteraction(int Count)

{

}

}

}

**Код программы для InfernalGrowthCell.cs:**

namespace GameLib.Ferm.Cells

{

public class InfernalGrowthCell : CellScript

{

public override void ChangeCellSprite()

{

if (playerChoise == gameObject.ObjectName)

{

gameObject.UpdateObjectSprite(gameObject, selectedCell);

}

else gameObject.UpdateObjectSprite(gameObject, unselectedCell);

}

public override void SeedCountInteraction(int Count)

{

gameObject.SetTextString(Count.ToString());

}

}

}

**Код программы для MoneyCell.cs:**

namespace GameLib.Ferm.Cells

{

public class MoneyCell : CellScript

{

public override void ChangeCellSprite()

{

}

public override void SeedCountInteraction(int Count)

{

gameObject.SetTextString(Count.ToString());

}

}

}

**Код программы для PumpkinCell.cs:**

namespace GameLib.Ferm.Cells

{

public class PumpkinCell : CellScript

{

public override void ChangeCellSprite()

{

if (playerChoise == gameObject.ObjectName)

{

gameObject.UpdateObjectSprite(gameObject, selectedCell);

}

else gameObject.UpdateObjectSprite(gameObject, unselectedCell);

}

public override void SeedCountInteraction(int Count)

{

gameObject.SetTextString(Count.ToString());

}

}

}

**Код программы для ReedCell.cs:**

namespace GameLib.Ferm.Cells

{

public class ReedCell : CellScript

{

public override void ChangeCellSprite()

{

if (playerChoise == gameObject.ObjectName)

{

gameObject.UpdateObjectSprite(gameObject, selectedCell);

}

else gameObject.UpdateObjectSprite(gameObject, unselectedCell);

}

public override void SeedCountInteraction(int Count)

{

gameObject.SetTextString(Count.ToString());

}

}

}

**Код программы для SunFlowerCell.cs:**

namespace GameLib.Ferm.Cells

{

public class SunFlowerCell : CellScript

{

public override void ChangeCellSprite()

{

if (playerChoise == gameObject.ObjectName)

{

gameObject.UpdateObjectSprite(gameObject, selectedCell);

}

else gameObject.UpdateObjectSprite(gameObject, unselectedCell);

}

public override void SeedCountInteraction(int Count)

{

gameObject.SetTextString(Count.ToString());

}

}

}

**Код программы для WateringCell.cs:**

namespace GameLib.Ferm.Cells

{

public class WateringCell : CellScript

{

public override void ChangeCellSprite()

{

if (playerChoise == gameObject.ObjectName)

{

gameObject.UpdateObjectSprite(gameObject, selectedCell);

}

else gameObject.UpdateObjectSprite(gameObject, unselectedCell);

}

public override void SeedCountInteraction(int Count)

{

gameObject.SetTextString(Count.ToString());

}

}

}

**Код программы для WheatCell.cs:**

namespace GameLib.Ferm.Cells

{

internal class WheatCell : CellScript

{

public override void ChangeCellSprite()

{

if (playerChoise == gameObject.ObjectName)

{

gameObject.UpdateObjectSprite(gameObject, selectedCell);

}

else gameObject.UpdateObjectSprite(gameObject, unselectedCell);

}

public override void SeedCountInteraction(int Count)

{

gameObject.SetTextString(Count.ToString());

}

}

}

**Код программы для DelayTime.cs:**

using System;

using System.Diagnostics;

namespace GameLib.Ferm.Delay

{

public class DelayTime

{

public float Delay { get; private set; }

public float CurrentTime { get; private set; }

private long previousTicks;

private float DeltaTime { get; set; }

private float \_delayLong { get; set; }

private Stopwatch watch;

public DelayTime(float delay)

{

watch = new Stopwatch();

Delay = delay;

DelayReset();

}

public void DeleyUpdate()

{

long ticks = watch.Elapsed.Ticks;

CurrentTime = (float)ticks / TimeSpan.TicksPerSecond;

DeltaTime = (float)(ticks - previousTicks) / TimeSpan.TicksPerSecond;

previousTicks = ticks;

}

public void DelayReset()

{

watch.Reset();

watch.Start();

previousTicks = watch.Elapsed.Ticks;

}

public bool Opportunity()

{

if (CurrentTime <= Delay)

return false;

else return true;

}

public void SetDelay(float delay)

{

Delay = delay;

}

}

}

**Код программы для GrowScript.cs:**

using EngineLib.ObjectComponents;

using GameLib.Game;

using EngineLib.EngineComponents;

using GameLib.Ferm.GrowScripts.DelayGrow;

namespace GameLib.Ferm.GrowScripts

{

public abstract class GrowScript : ScriptObject

{

public Inventory inventoryInstance { get; private set; }

protected Player playerInstance { get; private set; }

protected Fermas fermasInstance { get; set; }

protected string playerChoice { get; private set; }

private GrowDelay growDelay { get; set; }

public override void Start()

{

playerInstance = Player.staticPlayerInstatnce;

inventoryInstance = Player.staticInventoryInstance;

fermasInstance = Fermas.Instance;

playerChoice = Player.choiseCell;

growDelay = new GrowDelay();

}

public override void Update()

{

playerChoice = Player.choiseCell;

if (GrowDelay.GrowOpportunity())

{

if (Input.GatPozitionMove(playerInstance.GetControl().Space))

{

GameObject IntercectionObject = gameObject.Collider.TagedGameObject("Player");

if (IntercectionObject != null)

{

PlayerInteraction(gameObject);

}

}

}

}

public abstract void PlayerInteraction(GameObject gameObject);

}

}

**Код программы для LandScript.cs:**

using GameLib.Ferm.SeedFactories;

using EngineLib.ObjectComponents;

using SharpDX;

namespace GameLib.Ferm.GrowScripts

{

public class LandScript : GrowScript

{

public override void PlayerInteraction(GameObject gameObject)

{

if (inventoryInstance.GetComponentsCount("Pumpkin") > 0 && playerChoice == "Pumpkin")

{

if (gameObject.ChildGameobject == null)

{

PumpkinFactory pumpkinFactory = new PumpkinFactory();

GameObject seedObject = pumpkinFactory.CreateSeed(gameObject.Transform.Position, new Vector2(0f, -0.1f));

gameObject.ChildGameobject = seedObject;

inventoryInstance.AddCmponentsCount(seedObject.ObjectName);

fermasInstance.AddObjectOnScene(seedObject);

}

}

if (inventoryInstance.GetComponentsCount("Wheat") > 0 && playerChoice == "Wheat")

{

if (gameObject.ChildGameobject == null)

{

WheatFactory wheatFactory = new WheatFactory();

GameObject seedObject = wheatFactory.CreateSeed(gameObject.Transform.Position, new Vector2(0f, -0.2f));

gameObject.ChildGameobject = seedObject;

inventoryInstance.AddCmponentsCount(seedObject.ObjectName);

fermasInstance.AddObjectOnScene(seedObject);

}

}

if (inventoryInstance.GetComponentsCount("SunFlower") > 0 && playerChoice == "SunFlower")

{

if (gameObject.ChildGameobject == null)

{

SunFlowerFactory sunFlowerFactory = new SunFlowerFactory();

GameObject seedObject = sunFlowerFactory.CreateSeed(gameObject.Transform.Position, new Vector2(0f, -0.2f));

gameObject.ChildGameobject = seedObject;

inventoryInstance.AddCmponentsCount(seedObject.ObjectName);

fermasInstance.AddObjectOnScene(seedObject);

}

}

}

}

}

**Код программы для SendScript.cs:**

using GameLib.Ferm.SeedFactories;

using EngineLib.ObjectComponents;

using SharpDX;

namespace GameLib.Ferm.GrowScripts

{

public class SendScript : GrowScript

{

public override void PlayerInteraction(GameObject gameObject)

{

if (inventoryInstance.GetComponentsCount("Reed") > 0 && playerChoice == "Reed")

{

if (gameObject.ChildGameobject == null)

{

ReedFactory reedFactory = new ReedFactory();

GameObject seedObject = reedFactory.CreateSeed(gameObject.Transform.Position, new Vector2(0f, 0f));

gameObject.ChildGameobject = seedObject;

inventoryInstance.AddCmponentsCount(seedObject.ObjectName);

fermasInstance.AddObjectOnScene(seedObject);

}

}

}

}

}

**Код программы для ShowerSandScript.cs:**

using GameLib.Ferm.SeedFactories;

using EngineLib.ObjectComponents;

using SharpDX;

namespace GameLib.Ferm.GrowScripts

{

public class ShowerSandScript : GrowScript

{

public override void PlayerInteraction(GameObject gameObject)

{

if (inventoryInstance.GetComponentsCount("InfernalGrowth") > 0 && playerChoice == "InfernalGrowth")

{

if (gameObject.ChildGameobject == null)

{

InfernalGrowthFactory infernalGrowthFactory = new InfernalGrowthFactory();

GameObject seedObject = infernalGrowthFactory.CreateSeed(gameObject.Transform.Position, new Vector2(0f, -0.1f));

gameObject.ChildGameobject = seedObject;

inventoryInstance.AddCmponentsCount(seedObject.ObjectName);

fermasInstance.AddObjectOnScene(seedObject);

}

}

}

}

}

**Код программы для SeedFactory.cs:**

using EngineLib.ObjectComponents;

using SharpDX;

namespace GameLib.Ferm.SeedFactories

{

public abstract class SeedFactory

{

public abstract GameObject CreateSeed(Vector2 position, Vector2 offset = new Vector2());

protected static Vector2 GetOffsetVector(Vector2 position,Vector2 offset)

{

if(offset != null)

return position + offset;

else return position;

}

}

}

**Код программы для PumpkinFactory.cs:**

using EngineLib.ObjectComponents;

using SharpDX;

using EngineLib.EngineComponents;

using GameLib.Ferm.Seeds;

namespace GameLib.Ferm.SeedFactories

{

public class PumpkinFactory : SeedFactory

{

public override GameObject CreateSeed(Vector2 position, Vector2 offset = new Vector2())

{

GameObject gameObject = new GameObject();

position = GetOffsetVector(position, offset);

gameObject.ObjectComponentsInit("Pumpkin", new Transform(position, new Size2F(1f, 1f)),

new Sprite(RenderSystem.LoadBitmap(@"Ferm\Sprites\SmallSeeds\SmallPumpkin.png")),new Collider(gameObject, new Size2F(0.7f, 0.7f), new Vector2(-0.15f, 0.5f)));

gameObject.ScriptInit(new PumpkinSeed());

return gameObject;

}

}

}

**Код программы для InfernalGrowthFactory.cs:**

using EngineLib.ObjectComponents;

using GameLib.Ferm.Seeds;

using SharpDX;

using EngineLib.EngineComponents;

namespace GameLib.Ferm.SeedFactories

{

public class InfernalGrowthFactory : SeedFactory

{

public override GameObject CreateSeed(Vector2 position, Vector2 offset = new Vector2())

{

GameObject gameObject = new GameObject();

position = GetOffsetVector(position, offset);

gameObject.ObjectComponentsInit("InfernalGrowth", new Transform(position, new Size2F(1f, 1f)), new Sprite(RenderSystem.LoadBitmap(@"Ferm\Sprites\SmallSeeds\SmallInfernalGrowth.png")), new Collider(gameObject, new Size2F(0.5f, 0.5f)));

PumpkinSeed pumpkinSeed = new PumpkinSeed();

InfernalGrowthSeed infernalGrowthSeed = new InfernalGrowthSeed();

infernalGrowthSeed.SetDecoratedSeed(pumpkinSeed);

gameObject.ScriptInit(infernalGrowthSeed);

return gameObject;

}

}

}

**Код программы для ReedFactory.cs:**

using EngineLib.ObjectComponents;

using EngineLib.EngineComponents;

using GameLib.Ferm.Seeds;

using SharpDX;

namespace GameLib.Ferm.SeedFactories

{

internal class ReedFactory : SeedFactory

{

public override GameObject CreateSeed(Vector2 position, Vector2 offset = new Vector2())

{

GameObject gameObject = new GameObject();

position = GetOffsetVector(position, offset);

gameObject.ObjectComponentsInit("Reed", new Transform(position, new Size2F(1f, 0.94f)), new Sprite(RenderSystem.LoadBitmap(@"Ferm\Sprites\SmallSeeds\SmallReed.png")), new Collider(gameObject, new Size2F(0.5f, 0.5f)));

PumpkinSeed pumpkinSeed = new PumpkinSeed();

ReedSeed reedSeed = new ReedSeed();

reedSeed.SetDecoratedSeed(pumpkinSeed);

gameObject.ScriptInit(reedSeed);

return gameObject;

}

}

}

**Код программы для SunFlowerFactory.cs:**

using EngineLib.ObjectComponents;

using GameLib.Ferm.Seeds;

using SharpDX;

using EngineLib.EngineComponents;

namespace GameLib.Ferm.SeedFactories

{

public class SunFlowerFactory : SeedFactory

{

public override GameObject CreateSeed(Vector2 position, Vector2 offset = new Vector2())

{

GameObject gameObject = new GameObject();

position = GetOffsetVector(position, offset);

gameObject.ObjectComponentsInit("SunFlower", new Transform(position, new Size2F(1f, 1f)), new Sprite(RenderSystem.LoadBitmap(@"Ferm\Sprites\SmallSeeds\SmallSunFlower.png")), new Collider(gameObject, new Size2F(0.5f, 0.5f)));

PumpkinSeed pumpkinSeed = new PumpkinSeed();

SunFlowerSeed sunFlowerSeed = new SunFlowerSeed();

sunFlowerSeed.SetDecoratedSeed(pumpkinSeed);

gameObject.ScriptInit(sunFlowerSeed);

return gameObject;

}

}

}

**Код программы для WheatFactory.cs:**

using EngineLib.ObjectComponents;

using EngineLib.EngineComponents;

using GameLib.Ferm.Seeds;

using SharpDX;

namespace GameLib.Ferm.SeedFactories

{

public class WheatFactory : SeedFactory

{

public override GameObject CreateSeed(Vector2 position, Vector2 offset = new Vector2())

{

GameObject gameObject = new GameObject();

position = GetOffsetVector(position, offset);

gameObject.ObjectComponentsInit("Wheat", new Transform(position, new Size2F(1f, 1f)), new Sprite(RenderSystem.LoadBitmap(@"Ferm\Sprites\SmallSeeds\SmallWheat.png")), new Collider(gameObject, new Size2F(0.5f, 0.5f)));

PumpkinSeed pumpkinSeed = new PumpkinSeed();

WheatSeed wheatSeed = new WheatSeed();

wheatSeed.SetDecoratedSeed(pumpkinSeed);

gameObject.ScriptInit(wheatSeed);

return gameObject;

}

}

}

**Код программы для SeedTimeGrowing.cs:**

using System;

using System.Diagnostics;

namespace GameLib.Ferm.Seeds.GrowingTime

{

public class SeedTimeGrowing

{

private const float \_DAY\_LONG = 5f;

public int DayGrowing { get; private set; }

private float CurrentTime { get; set; }

private long previousTicks { get; set; }

private static float DeltaTime { get; set; }

private Stopwatch watch { get; set; }

public SeedTimeGrowing()

{

watch = new Stopwatch();

ResetSeedTime();

}

public void SeedTimeUpdate()

{

long ticks = watch.Elapsed.Ticks;

CurrentTime = (float)ticks / TimeSpan.TicksPerSecond;

DeltaTime = (float)(ticks - previousTicks) / TimeSpan.TicksPerSecond;

previousTicks = ticks;

int dayToadd = (int)(CurrentTime / \_DAY\_LONG);

if (dayToadd == 1 || dayToadd == 2)

{

DayGrowing++;

ResetSeedTime();

}

}

public void ResetSeedTime()

{

watch.Reset();

watch.Start();

previousTicks = watch.Elapsed.Ticks;

}

}

}

**Код программы для Seed.cs:**

using System;

using EngineLib.ObjectComponents;

using EngineLib.EngineComponents;

using GameLib.Ferm.Seeds.GrowingTime;

using GameLib.Ferm.GrowScripts.DelayGrow;

namespace GameLib.Ferm.Seeds

{

public enum GrowStatut

{

Grow,

Ready

}

public abstract class Seed : ScriptObject

{

public abstract int GrowTime { get; }

public int TimeChanger { get; private set; }

protected Fermas instance { get; private set; }

protected Sprite seedReadySprite { get; private set; }

public GrowStatut growStatut { get; private set; }

protected static Random random { get; private set; }

private SeedTimeGrowing seedGrowing { get; set; }

protected abstract int SeedCount{ get; }

static Seed()

{

random = new Random();

}

public override void Start()

{

instance = Fermas.Instance;

seedGrowing = new SeedTimeGrowing();

seedGrowing.ResetSeedTime();

seedReadySprite = new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Seeds\{gameObject.ObjectName}.png"));

growStatut = GrowStatut.Grow;

GrowDelay.DelayReset();

}

public override void Update()

{

GrowDelay.DeleyUpdate();

seedGrowing.SeedTimeUpdate();

GrowingProcess();

}

public void ChangeGrowTime()

{

if(GrowTime > 0)

TimeChanger++;

}

public virtual int GetRandomSeedCount()

{

return random.Next(0, 2);

}

public void GrowingProcess()

{

if(seedGrowing.DayGrowing == GrowTime - TimeChanger && growStatut == GrowStatut.Grow)

{

gameObject.UpdateObjectSprite(gameObject, seedReadySprite);

growStatut = GrowStatut.Ready;

}

}

public int GetSeedCount()

{

return SeedCount;

}

}

}

**Код программы для SeedDecorator.cs:**

namespace GameLib.Ferm.Seeds

{

public class SeedDecorator : Seed

{

public override int GrowTime => decoratedSeed.GrowTime;

protected override int SeedCount => decoratedSeed.GetSeedCount();

protected Seed decoratedSeed { get; set; }

public void SetDecoratedSeed(Seed seed)

{

decoratedSeed = seed;

}

}

}

**Код программы для PumpkinSeed.cs:**

namespace GameLib.Ferm.Seeds

{

public class PumpkinSeed : Seed

{

public override int GrowTime => 3;

protected override int SeedCount => GetRandomSeedCount() +1;

}

}

**Код программы для InfernalGrowthSeed.cs:**

namespace GameLib.Ferm.Seeds

{

public class InfernalGrowthSeed : SeedDecorator

{

public override int GrowTime => decoratedSeed.GrowTime + 5;

protected override int SeedCount => decoratedSeed.GetSeedCount();

}

}

**Код программы для ReedSeed.cs:**

namespace GameLib.Ferm.Seeds

{

public class ReedSeed : SeedDecorator

{

public override int GrowTime => decoratedSeed.GrowTime + 2;

protected override int SeedCount => decoratedSeed.GetSeedCount() + 2;

}

}

**Код программы для SunFlowerSeed.cs:**

namespace GameLib.Ferm.Seeds

{

public class SunFlowerSeed : SeedDecorator

{

public override int GrowTime => decoratedSeed.GrowTime + 1;

protected override int SeedCount => decoratedSeed.GetSeedCount() + 1;

}

}

**Код программы для WheatSeed.cs:**

namespace GameLib.Ferm.Seeds

{

public class WheatSeed : SeedDecorator

{

public override int GrowTime => decoratedSeed.GrowTime - 1;

protected override int SeedCount => decoratedSeed.GetSeedCount() + 2;

}

}

**Код программы для ToolEffectFactory.cs:**

using EngineLib.ObjectComponents;

using SharpDX;

namespace GameLib.Ferm.ToolEffects.ToolsEffectsFactory

{

public class ToolEffectFactory

{

public virtual GameObject CreateToolEffect(GameObject toolObject, ToolEffect toolEffect)

{

GameObject effectObject = new GameObject();

effectObject.ObjectComponentsInit(toolObject.ObjectName + "Effect", new Transform(new Vector2(0f, 0f)));

effectObject.ParentGameobject = toolObject;

toolObject.ChildGameobject = effectObject;

effectObject.ScriptInit(toolEffect);

return effectObject;

}

}

}

**Код программы для ToolEffect.cs:**

using System;

using EngineLib.ObjectComponents;

using GameLib.Game;

using GameLib.Ferm.Delay;

namespace GameLib.Ferm.ToolEffects

{

public abstract class ToolEffect : ScriptObject

{

protected Inventory inventoryInstance { get; private set; }

protected Random random { get; private set; }

protected abstract bool IsActive { get; set; }

protected abstract DelayTime delayTime { get; set; }

protected abstract float effectTime { get; set; }

protected abstract float currentEffectTime { get; set; }

public override void Start()

{

inventoryInstance = Player.staticInventoryInstance;

random = new Random();

ToolEffectDelayInit();

}

public override void Update()

{

Interaction();

}

public abstract void ActivateEffect();

public abstract void ToolEffectDelayInit();

protected abstract void DeactivateEffect();

public abstract void CheckActivate();

public abstract void Interaction();

}

}

**Код программы для BasketToolEffect.cs:**

using GameLib.Ferm.Delay;

using GameLib.Game;

using EngineLib.EngineComponents;

namespace GameLib.Ferm.ToolEffects

{

public class BasketToolEffect : ToolEffect

{

protected override DelayTime delayTime { get; set; }

protected override bool IsActive { get; set; }

protected override float effectTime { get; set; }

protected override float currentEffectTime { get; set; }

public bool isAnimated { get; set; } = false;

public override void ActivateEffect()

{

Player.staticInventoryInstance.RemoveComponentsCount("Watering", random.Next(1, 2));

}

protected override void DeactivateEffect()

{

}

public override void Interaction()

{

if (Time.CurrentTime < currentEffectTime + Time.DeltaTime)

Player.staticPlayerInstatnce.SetAddEffect(true);

else

isAnimated = false;

if (IsActive)

{

ActivateEffect();

}

if(!isAnimated)

{

Player.staticPlayerInstatnce.SetAddEffect(false);

}

IsActive = false;

}

public override void CheckActivate()

{

int randomValue = random.Next(1, 100);

if(randomValue < 30)

{

currentEffectTime = Time.CurrentTime + effectTime;

isAnimated = true;

IsActive = true;

}

}

public override void ToolEffectDelayInit()

{

effectTime = 1.5f;

delayTime = null;

}

}

}

**Код программы для WateringToolEffect.cs:**

using GameLib.Ferm.Delay;

using GameLib.Game;

using EngineLib.EngineComponents;

using GameLib.Ferm.GrowScripts.DelayGrow;

using GameLib.Ferm.Tools;

namespace GameLib.Ferm.ToolEffects

{

public class WateringToolEffect : ToolEffect

{

protected override DelayTime delayTime { get; set ; }

protected override bool IsActive { get; set; }

protected override float currentEffectTime { get; set; }

protected override float effectTime { get; set; }

public override void ActivateEffect()

{

Player.staticPlayerInstatnce.SetSpeed(6);

}

public override void CheckActivate()

{

int randomValue = random.Next(1, 100);

if (randomValue < 40)

{

IsActive = true;

currentEffectTime = Time.CurrentTime + effectTime;

}

else

{

IsActive = false;

}

}

public override void Interaction()

{

if(IsActive)

{

ActivateEffect();

GrowDelay.Delay = 0.3f;

SetBasketDelay(0.3f);

Player.staticPlayerInstatnce.SetSpeedEffect(true);

}

if(Time.CurrentTime >= currentEffectTime)

{

DeactivateEffect();

GrowDelay.Delay = 0.5f;

SetBasketDelay(0.5f);

Player.staticPlayerInstatnce.SetSpeedEffect(false);

}

}

public override void ToolEffectDelayInit()

{

effectTime = 10.0f;

}

public void SetBasketDelay(float delay)

{

(Player.staticInventoryInstance.GetToolObject("Basket").scriptObject as Tool).GetDelayTime().SetDelay(delay);

}

protected override void DeactivateEffect()

{

Player.staticPlayerInstatnce.SetSpeed(4);

}

}

}

**Код программы для ToolFactory.cs:**

using EngineLib.ObjectComponents;

using SharpDX;

using EngineLib.EngineComponents;

namespace GameLib.Ferm.Tools.ToolsFactory

{

public class ToolFactory

{

public virtual GameObject CreateTool(Vector2 position, string ToolName, Tool tool)

{

GameObject basketGameObject = new GameObject();

basketGameObject.ObjectComponentsInit(ToolName, new Transform(position),

new Sprite(RenderSystem.LoadBitmap($@"Ferm\Sprites\Tools\{ToolName}.png")),

new Collider(basketGameObject, new Size2F(0.7f, 0.7f), new Vector2(-0.15f, 0.5f)));

basketGameObject.ScriptInit(tool);

return basketGameObject;

}

}

}

**Код программы для Tool.cs:**

using System;

using EngineLib.ObjectComponents;

using GameLib.Game;

using EngineLib.EngineComponents;

using GameLib.Ferm.Delay;

namespace GameLib.Ferm.Tools

{

public abstract class Tool : ScriptObject

{

protected Player playerInstance { get; set; }

protected Inventory inventoryInstance { get; set; }

protected Fermas fermasInstance { get; set; }

protected string PLayerChois { get; set; }

protected Random random { get; private set; }

protected abstract DelayTime delayTime { get; set; }

public override void Start()

{

PLayerChois = Player.choiseCell;

playerInstance = Player.staticPlayerInstatnce;

inventoryInstance = Player.staticInventoryInstance;

fermasInstance = Fermas.Instance;

random = new Random();

ToolDelayInit();

}

public override void Update()

{

gameObject.Transform.Position = gameObject.ParentGameobject.Transform.Position;

PLayerChois = Player.choiseCell;

delayTime?.DeleyUpdate();

if (Input.GatPozitionMove(playerInstance.GetControl().Space))

{

PlayerInteraction();

}

}

public DelayTime GetDelayTime()

{

return delayTime;

}

public abstract void PlayerInteraction();

public abstract void ToolDelayInit();

public abstract void ToolDelayReset();

}

}

**Код программы для BasketTool.cs:**

using EngineLib.ObjectComponents;

using GameLib.Ferm.Seeds;

using GameLib.Ferm.GrowScripts;

using GameLib.Ferm.Delay;

using GameLib.Ferm.ToolEffects;

namespace GameLib.Ferm.Tools

{

public class BasketTool : Tool

{

protected override DelayTime delayTime { get; set; }

public override void PlayerInteraction()

{

if(gameObject.Collider.CheckIntersection("Pumpkin", "Wheat", "SunFlower", "Reed", "InfernalGrowth") && PLayerChois == "Basket" && delayTime.Opportunity())

{

GameObject landObject = gameObject.Collider.TagedGameObject("Land","Send", "ShowerSand");

if (landObject != null)

{

if (landObject.ChildGameobject != null && (landObject.ChildGameobject.scriptObject as Seed).growStatut != GrowStatut.Grow)

{

fermasInstance.RemoveObjectOnScene(landObject.ChildGameobject);

(landObject.scriptObject as GrowScript).inventoryInstance.RemoveComponentsCount(landObject.ChildGameobject.ObjectName,

(landObject.ChildGameobject.scriptObject as Seed).GetSeedCount());

switch(landObject.ChildGameobject.ObjectName)

{

case "Pumpkin":

{

playerInstance.ChangeCoinsValue(6);

break;

}

case "Wheat":

{

playerInstance.ChangeCoinsValue(2);

break;

}

case "SunFlower":

{

playerInstance.ChangeCoinsValue(5);

break;

}

case "Reed":

{

playerInstance.ChangeCoinsValue(4);

break;

}

case "InfernalGrowth":

{

playerInstance.ChangeCoinsValue(12);

break;

}

}

landObject.ChildGameobject = null;

(gameObject.ChildGameobject.scriptObject as BasketToolEffect).CheckActivate();

ToolDelayReset();

}

}

}

}

public override void ToolDelayInit()

{

delayTime = new DelayTime(0.5f);

}

public override void ToolDelayReset()

{

delayTime.DelayReset();

}

}

}

**Код программы для WateringTool.cs:**

using EngineLib.ObjectComponents;

using GameLib.Ferm.Seeds;

using GameLib.Ferm.GrowScripts;

using GameLib.Ferm.Delay;

using GameLib.Ferm.ToolEffects;

namespace GameLib.Ferm.Tools

{

class WateringTool : Tool

{

protected override DelayTime delayTime { get; set; }

public override void PlayerInteraction()

{

if (gameObject.Collider.CheckIntersection("Pumpkin", "Wheat", "SunFlower", "Reed", "InfernalGrowth") && PLayerChois == "Watering" && inventoryInstance.GetComponentsCount(gameObject.ObjectName) > 0

&& delayTime.Opportunity())

{

GameObject landObject = gameObject.Collider.TagedGameObject("Land", "Send", "ShowerSand");

if (landObject != null)

{

if (landObject.ChildGameobject != null && (landObject.ChildGameobject.scriptObject as Seed).growStatut == GrowStatut.Grow)

{

(landObject.ChildGameobject.scriptObject as Seed).ChangeGrowTime();

(gameObject.ChildGameobject.scriptObject as WateringToolEffect).CheckActivate();

(landObject.scriptObject as GrowScript).inventoryInstance.RemoveComponentsObject(gameObject, -1);

ToolDelayReset();

}

}

}

}

public override void ToolDelayInit()

{

delayTime = new DelayTime(1.5f);

}

public override void ToolDelayReset()

{

delayTime.DelayReset();

}

}

}

**Код программы для ApplicationsPicture.cs:**

using System.Drawing;

using System.Windows.Forms;

namespace GameApp.FormInit

{

public class ApplicationsPicture

{

private Form1 \_form { get;}

public ApplicationsPicture(Form1 form)

{

\_form = form;

}

public void SetBackGround()

{

\_form.BackgroundImage = new Bitmap($@"Resources\BackGroundPicture\BackGroundForm.jpg");

}

public void SetPlayButton(Button button)

{

button.Image = new Bitmap($@"Resources\GameStartPicture\Play.png");

button.BackColor = Color.Transparent;

}

public void SetResultPictureBox(PictureBox pictureBox, string ResultName)

{

pictureBox.Image = new Bitmap($@"Resources\GameResultPictures\Result{ResultName}.png");

pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;

pictureBox.BackColor = Color.Transparent;

}

public void SetButtonSkin(Button button, string skinName)

{

button.Image = new Bitmap($@"Resources\GameSkinsButton\{skinName}.png");

}

public void SetSkinPictureBox(PictureBox pictureBox, string SkinName)

{

pictureBox.BackgroundImage = Image.FromFile($@"Resources\PlayerPictures\SkinBackGround{SkinName}.png");

pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;

}

public void SetSkinPictureBoxBackGround(PictureBox pictureBox)

{

pictureBox.BackgroundImage = Image.FromFile($@"Resources\SkinsBackground\SkinsBackGround.png");

pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;

}

public void SetChangeValueButtonSkin(Button button, string direction)

{

button.BackColor = Color.Transparent;

button.BackgroundImage = Image.FromFile($@"Resources\ChangeValue\Land{direction}.png");

}

public void SetGameTargetsPictureBox(PictureBox pictureBox, string targetName)

{

pictureBox.Image = new Bitmap($@"Resources\ValueTarget\{targetName}.png");

pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;

pictureBox.BackColor = Color.Transparent;

}

}

}

**Код программы для Form1.cs:**

using System;

using System.Drawing;

using System.Windows.Forms;

using EngineLib.EngineComponents;

using GameLib.Ferm;

using GameLib.Game;

using GameApp.FormInit;

namespace GameApp

{

public partial class Form1 : Form

{

private ApplicationRendering GameApp;

private Scene Scene { get; set; }

private const int \_DEFAULTE\_DAYS\_VALUE = 25;

private const int \_DEFAULTE\_COINS\_VALUE = 500;

private int currentDaysValue { get; set; } = \_DEFAULTE\_DAYS\_VALUE;

private int currentCoinsValue { get; set; } = \_DEFAULTE\_COINS\_VALUE;

private ApplicationsPicture pictureApplication { get; set; }

public Form1()

{

InitializeComponent();

FormInit();

GameIvents.EndGame += EndGame;

this.Text = "Launch";

}

private void button1\_Click(object sender, EventArgs e)

{

//this.Hide();

StartPlayButton.Enabled = false;

GameApp = new ApplicationRendering();

Scene = new Fermas(currentDaysValue, currentCoinsValue);

GameApp.SetScene(Scene);

GameApp.Run();

}

private void FormInit()

{

pictureApplication = new ApplicationsPicture(this);

pictureApplication.SetBackGround();

pictureApplication.SetPlayButton(StartPlayButton);

pictureApplication.SetResultPictureBox(pictureBoxResult, "Player");

pictureApplication.SetButtonSkin(SpeedImagebutton, "Speed");

pictureApplication.SetButtonSkin(PlayerImagebutton, "Player");

pictureApplication.SetButtonSkin(BasketImagebutton, "Basket");

pictureApplication.SetSkinPictureBoxBackGround(SkinsPictureBox);

pictureApplication.SetChangeValueButtonSkin(DaysRightButton, "Right");

pictureApplication.SetChangeValueButtonSkin(DaysLeftButton, "Left");

pictureApplication.SetChangeValueButtonSkin(CoinsRightButton, "Right");

pictureApplication.SetChangeValueButtonSkin(CoinsLeftButton, "Left");

pictureApplication.SetGameTargetsPictureBox(pictureBoxDays, "Days");

pictureApplication.SetGameTargetsPictureBox(pictureBoxCoins, "Coins");

DaysLabel.BackColor = Color.Transparent;

DaysLabel.ForeColor = Color.OrangeRed;

CoinsLabel.BackColor = Color.Transparent;

CoinsLabel.ForeColor = Color.LightYellow;

DaysLabel.Text = \_DEFAULTE\_DAYS\_VALUE.ToString();

CoinsLabel.Text = \_DEFAULTE\_COINS\_VALUE.ToString();

DaysResultLabel.BackColor = Color.Transparent;

CoinsResultLabel.BackColor = Color.Transparent;

}

private void EndGame(int Days, int Coins)

{

this.Visible = true;

GameApp.Dispose();

GameApp = null;

DaysResultLabel.Text = "Дней пройдено: " + Days;

CoinsResultLabel.Text = "Монет заработано: " + Coins;

if(Coins >= currentCoinsValue)

pictureApplication.SetResultPictureBox(pictureBoxResult, "Win");

else

pictureApplication.SetResultPictureBox(pictureBoxResult, "Lost");

}

private void PlayerImagebutton\_Click(object sender, EventArgs e)

{

pictureApplication.SetSkinPictureBox(SkinsPictureBox, "Player");

}

private void SpeedImagebutton\_Click(object sender, EventArgs e)

{

pictureApplication.SetSkinPictureBox(SkinsPictureBox, "Speed");

}

private void BasketImagebutton\_Click(object sender, EventArgs e)

{

pictureApplication.SetSkinPictureBox(SkinsPictureBox, "Basket");

}

private void DaysRightButton\_Click(object sender, EventArgs e)

{

currentDaysValue = TargetValueUpdate('+', currentDaysValue);

DaysLabel.Text = currentDaysValue.ToString();

}

private void DaysLeftButton\_Click(object sender, EventArgs e)

{

currentDaysValue = TargetValueUpdate('-', currentDaysValue);

DaysLabel.Text = currentDaysValue.ToString();

}

private void CoinsLeftButton\_Click(object sender, EventArgs e)

{

currentCoinsValue = TargetValueUpdate('-', currentCoinsValue);

CoinsLabel.Text = currentCoinsValue.ToString();

}

private void CoinsRightButton\_Click(object sender, EventArgs e)

{

currentCoinsValue = TargetValueUpdate('+', currentCoinsValue);

CoinsLabel.Text = currentCoinsValue.ToString();

}

private int TargetValueUpdate(char sign ,int value)

{

if (value > 0 && value < 1000)

{

switch(sign)

{

case '+':

{

value++;

break;

}

case '-':

{

value--;

break;

}

}

}

else

return 1;

return value;

}

}

}

**Код программы для UnitTestCollider.cs:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using EngineLib.ObjectComponents;

using Transform = EngineLib.ObjectComponents.Transform;

using SharpDX;

namespace TestGameApp

{

[TestClass]

public class UnitTestCollider

{

[TestMethod]

public void TestCollider()

{

GameObject testGameObject1 = new GameObject();

testGameObject1.ObjectComponentsInit("object1", new Transform(new Vector2(1f, 1f), new Size2F(1f, 1f)), null, new Collider(testGameObject1, new Size2F(1f,1f)));

GameObject testGameObject2 = new GameObject();

testGameObject2.ObjectComponentsInit("object2", new Transform(new Vector2(1f, 3f), new Size2F(1f, 1f)), null, new Collider(testGameObject2, new Size2F(1f, 1f)));

Assert.IsFalse(testGameObject1.Collider.CheckIntersection("object2"));

testGameObject2.Transform.Position = new Vector2(1f, 1f);

Assert.IsTrue(testGameObject1.Collider.CheckIntersection("object2"));

}

}

}

**Код программы для UnitTestMovement.cs:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using EngineLib.ObjectComponents;

using Transform = EngineLib.ObjectComponents.Transform;

using SharpDX;

namespace TestGameApp

{

[TestClass]

public class UnitTestMovement

{

[TestMethod]

public void TestMovement()

{

GameObject gameObject = new GameObject();

gameObject.ObjectComponentsInit("object1", new Transform(new Vector2(1f, 1f), new Size2F(1f, 1f)));

Vector2 offset = new Vector2(1f, 1f);

gameObject.Transform.SetMovement(offset);

Vector2 expected = new Vector2(2f, 2f);

Vector2 actual = gameObject.Transform.Position;

Assert.AreEqual(expected, actual);

}

}

}

**Код программы для UnitTestDecorator.cs:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using GameLib.Ferm.Seeds;

namespace TestGameApp

{

[TestClass]

public class UnitTestDecorator

{

[TestMethod]

public void DecoratedReed()

{

PumpkinSeed pumpkinSeed = new PumpkinSeed();

ReedSeed reedSeed = new ReedSeed();

reedSeed.SetDecoratedSeed(pumpkinSeed);

int difference = reedSeed.GetGrowTime() - pumpkinSeed.GetGrowTime();

Assert.AreEqual(pumpkinSeed.GrowTime, reedSeed.GrowTime - difference );

}

[TestMethod]

public void DecoratedInfernalGrowth()

{

PumpkinSeed pumpkinSeed = new PumpkinSeed();

InfernalGrowthSeed infernalGrowthSeed = new InfernalGrowthSeed();

infernalGrowthSeed.SetDecoratedSeed(pumpkinSeed);

int difference = infernalGrowthSeed.GetGrowTime() - pumpkinSeed.GetGrowTime();

Assert.AreEqual(pumpkinSeed.GrowTime, infernalGrowthSeed.GrowTime - difference);

}

[TestMethod]

public void DecoratedWheat()

{

PumpkinSeed pumpkinSeed = new PumpkinSeed();

WheatSeed wheatSeed = new WheatSeed();

wheatSeed.SetDecoratedSeed(pumpkinSeed);

int difference = wheatSeed.GetGrowTime() - pumpkinSeed.GetGrowTime();

Assert.AreEqual(pumpkinSeed.GrowTime, wheatSeed.GrowTime - difference);

}

[TestMethod]

public void DecoratedSunFlower()

{

PumpkinSeed pumpkinSeed = new PumpkinSeed();

SunFlowerSeed sunFlowerSeed = new SunFlowerSeed();

sunFlowerSeed.SetDecoratedSeed(pumpkinSeed);

int difference = sunFlowerSeed.GetGrowTime() - pumpkinSeed.GetGrowTime();

Assert.AreEqual(pumpkinSeed.GrowTime, sunFlowerSeed.GrowTime - difference);

}

}

}

**Код программы для UnitTestSeedFactory.cs:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using EngineLib.ObjectComponents;

using GameLib.Ferm.SeedFactories;

using SharpDX;

namespace TestGameApp

{

[TestClass]

public class UnitTestSeedFactory

{

[TestMethod]

public void TestPumpkinFactory()

{

PumpkinFactory pumpkinFactory = new PumpkinFactory();

GameObject gameObject = pumpkinFactory.CreateSeed(new Vector2(1f,1f));

string excpectedSeed = "Pumpkin";

Assert.AreEqual(excpectedSeed, gameObject.ObjectName);

}

[TestMethod]

public void TestWheatFactory()

{

WheatFactory wheatFactory = new WheatFactory();

GameObject gameObject = wheatFactory.CreateSeed(new Vector2(1f, 1f));

string excpectedSeed = "Wheat";

Assert.AreEqual(excpectedSeed, gameObject.ObjectName);

}

[TestMethod]

public void TestReedFactory()

{

ReedFactory reedFactory = new ReedFactory();

GameObject gameObject = reedFactory.CreateSeed(new Vector2(1f, 1f));

string excpectedSeed = "Reed";

Assert.AreEqual(excpectedSeed, gameObject.ObjectName);

}

[TestMethod]

public void TestSunFlowerFactory()

{

SunFlowerFactory sunFlowerFactory = new SunFlowerFactory();

GameObject gameObject = sunFlowerFactory.CreateSeed(new Vector2(1f, 1f));

string excpectedSeed = "SunFlower";

Assert.AreEqual(excpectedSeed, gameObject.ObjectName);

}

[TestMethod]

public void TestInfernalGrowthFactory()

{

InfernalGrowthFactory infernalGrowthFactory = new InfernalGrowthFactory();

GameObject gameObject = infernalGrowthFactory.CreateSeed(new Vector2(1f, 1f));

string excpectedSeed = "InfernalGrowth";

Assert.AreEqual(excpectedSeed, gameObject.ObjectName);

}

}

}

# **ПРИЛОЖЕНИЕ Б**

(обязательное)

**Архитектура игрового приложения «Ферма»**

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Руководство пользователя**

Введение.

Разработанное программное приложение предназначено для запуска на ОС не ниже Windows 7. Игра обладает минимальным порогом вхождения для пользователей.

Игровое приложение обладает следующим функционалом:

– игровой уровень в виде фермы с элементами для взаимодействия, бонусами и инструментами;

– передвижение игрока;

Для использования программного приложения пользователь должен быть ознакомлен со следующими документами:

– настоящим руководством пользователя;

– правилами использования ЭВМ.

Назначение и условия применения.

Разработанное программное приложение предназначено для игры одного игрока. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходимо следующая конфигурация технических средств аппаратного обеспечения:

– центральный процессор Intel Core 2 Duo c тактовой частотой 2.30 МГц или более;

– наличие клавиатуры, мыши и цветного монитора SVGA с разрешением не менее 1280x720;

– операционная система Windows 7 и выше;

– 1024 Мб оперативной памяти.

Подготовка к работе.

Приложение запускается путём открытия файла GameApp.exe. Также на компьютере должны быть установлены драйвера для видеокарты. Если все инструкции соблюдены, и приложение не выдаёт никаких сообщений об ошибках, значит, программа работает исправно.

Описание операций.

При запуске приложения открывается интерфейс, который выводит изображение с информацией об игре и кнопкой запуска игры.

Цель игрока заключается в собирании урожая и получении за это денег (монет). В процессе игры можно передвигаться по горизонтали и вертикали.

Аварийные ситуации

Чтобы избежать ошибок при использовании программы, необходимо соблюдать порядок действий и условия пользования, описанные в пункте 3 данного руководства пользователя.

В случае непредвиденного «зависания» программы рекомендуется завершить процесс в диспетчере задач и запустить снова.

Рекомендации по освоению.

Заранее изучить работу с клавиатурой персонального компьютера. Запомнить расположение клавиш, отвечающих за игровое управление.

**ПРИЛОЖЕНИЕ Г**

(обязательное)

**Руководство программиста**

Назначения и условия применения программы.

Разработанное игровое приложение предназначено для игры между двумя игроками на одном компьютере. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

– центральный процессор Intel Core 2 Duo c тактовой частотой 2.30 МГц или более;

– наличие клавиатуры, мыши и цветного монитора SVGA с разрешением не менее 1280x720;

– операционная система Windows 7 и выше;

– 1024 Мб оперативной памяти.

1. Характеристики программы.

Для запуска приложения не требуется никаких дополнительных настроек. Для запуска решения необходима среда разработки Visual Studio с установленными фреймворком .NET и библиотекой SharpDX.

1. Обращение к программе.

Приложение запускается путём открытия файла GameApp.exe, находящегося в папке game.

1. Входные и выходные данные.

В данной программе в качестве входных данных используется ввод с клавиатуры кнопок управления игровым процессом. В качестве выходных выступает окно отображения игры.

1. Сообщения.

При окончании игры закрывается окно игры, после чего в интерфейсе пользователя черным цветом отображается статистика игрока (количество набранных монет и пройденных дней).

# **ПРИЛОЖЕНИЕ Д**

(обязательное)

**Руководство системного программиста**

1. Общие сведения о программе.

Разработанное игровое приложение предназначено для игры одного игрока на одном компьютере. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

* центральный процессор *Intel Core* 2 *Duo* c тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и цветного монитора *SVGA* с разрешением не менее 1280x720;
* операционная система *Windows* 7 и выше;
* 1024 Мб оперативной памяти.

1. Структура программы.

Игровое приложение логически можно разбить на несколько составляющих: игровой движок, содержащий средства работы с графикой, непосредственно логика игровых объектов и игрового процесса и графический интерфейс пользователя.

1. Настройка программы.

Для запуска приложения не требуется никаких дополнительных настроек. Для запуска решения необходима среда разработки *Visual Studio* с установленными фреймворком .*NET* и библиотекой *SharpDX.*

1. Проверка программы.

Для верификации программного средства были реализованы модульные тесты. Для запуска отладки в среде разработки *Visual Studio* необходимо запустить выполнение модульных тестов. По окончанию процесса отладки будет выдан отчет о результатах тестирования приложения.

1. Дополнительные возможности.

Приложение является узконаправленным и не имеет дополнительных возможностей.

1. Сообщение системному программисту.

Проект игрового движка можно использовать для разработки других игр. Проект игровой логики можно дополнить, добавив новые поляны, растения инструменты и элементы поля.